

DELPHI

程序员成长之路丛书

程序员成长攻略

蒙祖强 龚 涛 等编著

全套编程方法和技巧

基于.NET框架软件开发

Delphi程序员的首选

图文并茂 易于阅读

实例讲解 易于运用



中国水利水电出版社
www.waterpub.com.cn

程序员成长之路丛书

Delphi 程序员成长攻略

蒙祖强 龚 涛 等编著

中国水利水电出版社

内 容 提 要

本书从程序员成长的历练过程出发,由浅入深、全面系统地介绍了以 Delphi 2005 为工具的 Delphi 应用程序编程技能和开发方法。

全书分为 14 章,主要介绍了 Delphi 2005 集成开发环境 (IDE), Object Pascal 编程语言, Delphi 中消息和事件的处理机制, GUI 应用程序开发, 菜单的创建与设计, SQL 基础及 SQL Server 2000 数据库, 基于 BDE、ADO.NET、dbExpress、BDP 的数据库应用开发, 报表设计及数据输出, ASP.NET 应用程序开发, 以及 ASP.NET Web 服务应用程序开发, 并配以大量的开发实例。本书知识涵盖全面, 逻辑层次清楚, 图文并茂, 紧跟现代计算机应用技术的步伐, 是一本从事软件开发 (特别是数据库应用开发) 的优秀参考书。

本书内容丰富、实例详尽, 适合于具有一定编程经验的程序员、开发人员和 Delphi 爱好者使用, 也有助于具有丰富开发经验的系统分析员、系统测试员、企业 IT 经理等参考, 同时也是 Delphi 初学者迅速提高编程水平的一本好的参考书。

本书配有源代码, 读者可到中国水利水电出版社网站 (<http://www.waterpub.com.cn/sofdown>) 免费下载。

图书在版编目 (CIP) 数据

Delphi 程序员成长攻略 / 蒙祖强等编著. —北京: 中国水利水电出版社, 2007

(程序员成长之路丛书)

ISBN 978-7-5084-4284-6

I. D... II. 蒙... III. 软件工具—程序设计
IV.TP311.56

中国版本图书馆 CIP 数据核字 (2006) 第 149895 号

书 名	Delphi 程序员成长攻略
作 者	蒙祖强 龚 涛 等编著
出版 发行	中国水利水电出版社 (北京市三里河路 6 号 100044) 网址: www.waterpub.com.cn E-mail: mchannel@263.net (万水) sales@waterpub.com.cn 电话: (010) 63202266 (总机)、68331835 (营销中心)、82562819 (万水)
经 售	全国各地新华书店和相关出版物销售网点
排 版	北京万水电子信息有限公司
印 刷	北京蓝空印刷厂
规 格	787mm×1092mm 16 开本 30.5 印张 742 千字
版 次	2007 年 2 月第 1 版 2007 年 2 月第 1 次印刷
印 数	0001—4000 册
定 价	48.00 元

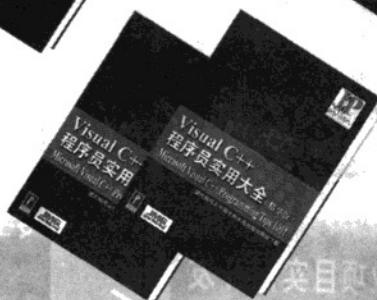
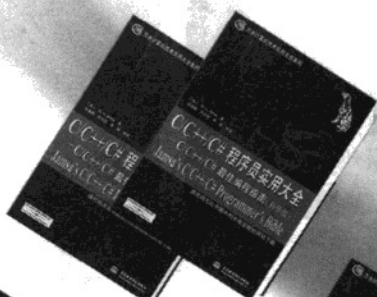
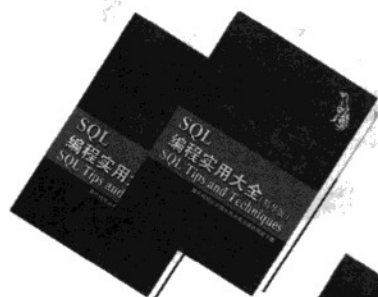
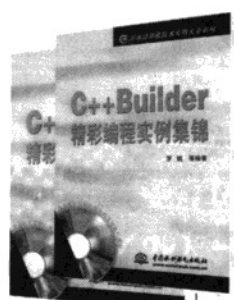
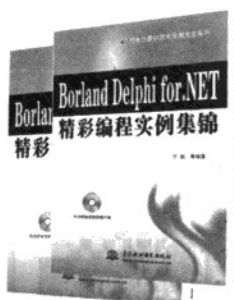
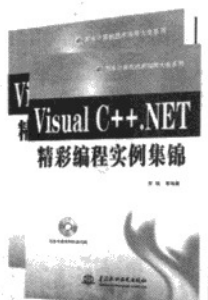
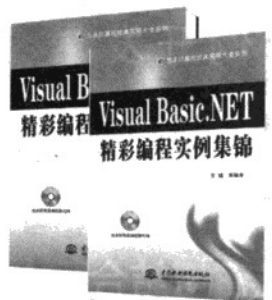
凡购买我社图书, 如有缺页、倒页、脱页的, 本社营销中心负责调换

版权所有·侵权必究



中国水利水电出版社
www.waterpub.com.cn

万水计算机技术实用大全系列



北京万水电子信息有限公司
Beijing Multi-Channel Electronic Information Co., Ltd.

地址: 北京市海淀区长春桥路5号新起点嘉园4号楼
1706室

电话: 010-82562819

传真: 010-82564371

邮编: 100089

E-mail: mchannel@263.net

NIIT资深开发人员编写

创新性的内容安排

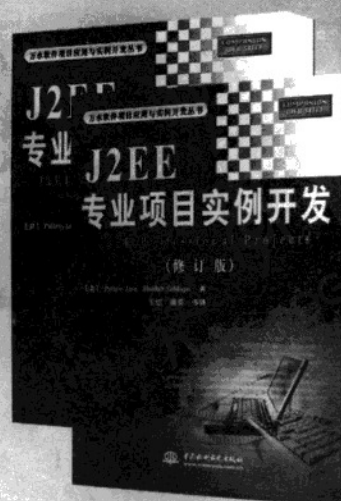
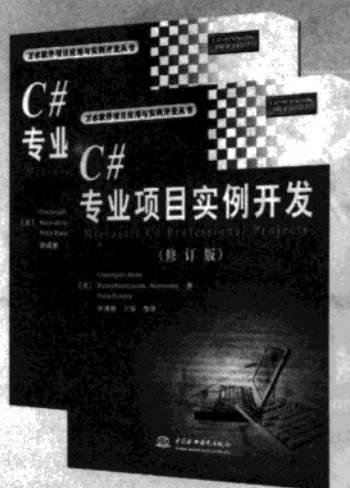
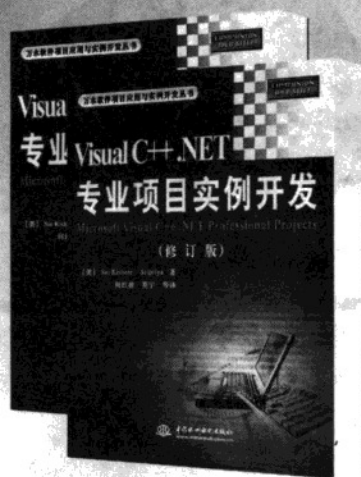
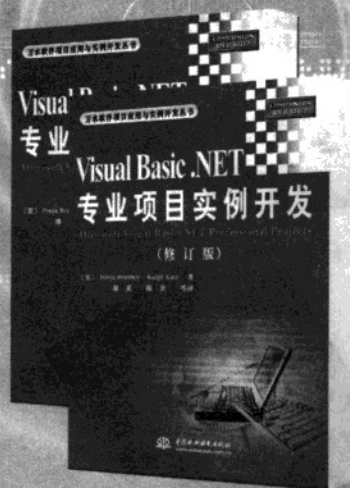
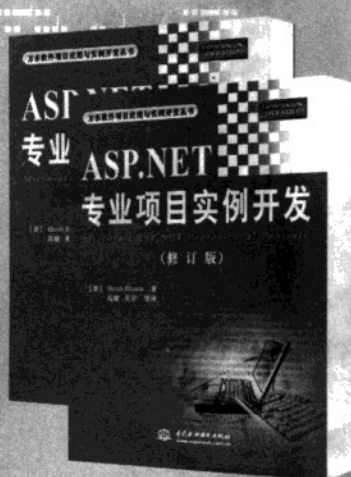
实用的案例开发

丰富的实例代码

全面的知识讲解

内容由浅入深 循序渐进

理论与实践密切结合



前 言

许多编程新手在成为程序员之前，总是充满希望和迷惑。程序员成长之路到底是怎样的？如何成为成功的程序员？对于我们老程序员来说，回顾往日奋斗的历程，更多的是感慨，心想如果当初有高手指点迷津，程序员的磨练过程中就可以少走许多弯路，少浪费许多时间，获得更大的成功。当初，很少有人提供这些经验，市面上也没有系统阐述此类经验的书籍，因此就有了借“程序员成长之路”丛书照亮年轻程序员成长之路的想法。幸运的是，这个想法得到了中国水利水电出版社的领导和编辑的肯定和支持，又有了中南大学一批志同道合的程序老手的合作，从而使“程序员成长之路”丛书诞生了。

这套丛书主要针对市面上应用较为广泛、实用价值高的编程语言，总结有关程序员的成长经验，以众多实例的形式展示编程技术提升的过程。这些常用编程技术包括 Visual C++、ASP.NET、Java、Delphi、C++ Builder、JSP、MATLAB、C、Web 等编程语言和工具，还包括数据库编程开发工具 DB2，以及面向对象编程的框架和方法。各种编程语言和工具好比“侠客”手中的“武器”，这套丛书好比是各种编程“大侠”展现其“武器”的绝技。十八般武器，各显神通。

Delphi 2005 是 Borland 公司目前推出的 Delphi 最新版本，它具有操作简单、执行效率高、功能强大等特点，是最容易入门的可视化编程工具之一。Delphi 8 和 Delphi 2005 是 Delphi 向 .NET 框架移植的版本。虽然 Borland 公司宣称“Delphi 8 是纯正的 .NET，也是纯正的 Delphi”，但在实际运用当中还存在不少不尽如人意的地方，以至于很多人还是停留在 Delphi 6/7 当中，更有甚者称之为“半成品”。也许正是出于对这方面的考虑，Borland 公司于 2004 年 11 月又推出了目前最新的 Delphi 版本——Delphi 2005。该版本对 Delphi 8 进行了多方面的改进，可算是真正的 Delphi .NET 版本。

作为基于 .NET 框架的软件开发工具，Delphi 2005 肯定是 Delphi 程序员的首选，因为基于 .NET 技术的软件开发方法肯定是今后软件开发的主流。Delphi 2005 有三个版本，即 Borland Delphi Architect、Borland Delphi Enterprise 和 Borland Delphi Professional。本书主要以 Borland Delphi Architect 为例，展开对基于 Delphi 2005 的软件应用开发的阐述，由此读者可以方便地引伸到其他版本的学习和应用开发当中。

本书共分 14 章，由浅入深、从易到难，着重介绍基于 Delphi 2005 软件开发的知识和实际编程的方法，体现了程序员成长的历练过程。

第 1 章介绍了学习 Delphi 的理由、Delphi 程序员的价值及 Delphi 程序员的编程规则。

第 2 章着重介绍 Delphi 2005 集成开发环境 (IDE) 及 Delphi 应用程序的一般创建方法。

第 3 章主要对 Object Pascal 程序设计的语法特征进行介绍，同时赋予大量而具体的例子，并对 Delphi 中基于 OOP 技术的编程方法进行了介绍。

第 4 章主要介绍 Delphi 中消息和事件的捕获、传递和处理机制。

第5章介绍如何在 Delphi 中开发 GUI 应用程序,着重介绍了常用 VCL 组件的使用方法和技巧,单文档和多文档应用程序的创建和设计原理,菜单及弹出式菜单的制作方法和编程技巧。

第6章着重介绍 SQL Server 2000 的使用方法,然后以此为平台详细介绍 SQL (结构查询语言)的语法。

第7章介绍 BDE 数据库开发技术,包括常用的 BDE 组件及相关数据库组件的属性和方法,然后介绍了数据库管理的基本操作,如数据浏览、查询设计、数据的插入、数据的删除、数据的修改等,最后用一个具体的例子对所学的内容进行了总结。

第8章主要介绍 .NET 技术之一——ADO.NET 技术,以及基于 ADO.NET 技术的数据库应用开发方法。

第9章对 dbExpress 技术及其应用进行较为全面的介绍,读者由此可以掌握基于 dbExpress 的跨平台数据库应用开发方法。

第10章介绍基于 BDP (Borland Data Provider) 的数据库应用开发。BDP 提供了一种高性能的数据访问架构和统一的编程模型,用 BDP 开发的程序易于发布,具有开发速度快、过程简洁等特点。

第11章介绍基于 Rave 组件的报表设计技术及数据输出方法。Delphi 2005 对报表设计提供了很好的支持,它把 Nevrona 公司开发的 Rave 组件嵌入其中,使得报表设计变得简便而快捷。

第12章介绍 ASP.NET 的新特点、ASP.NET 开发技术,以及基于 ASP.NET 技术的应用程序开发技能,最终让读者掌握 ASP.NET 应用程序开发技术。

第13章着重介绍 ASP.NET Web Services 应用的开发技术。随着 Internet 技术的迅猛发展和网上资源的日益丰富,如何把这些资源整合起来为特定的应用服务,是一个很现实的问题。Web Services (Web 服务)正是迎合了这种需要而不断得到发展的一种热门的 Web 开发技术。

第14章详细介绍基于 Delphi 2005 开发工具的数据库应用开发实例——学生信息管理系统的设计和开发方法,其中还涉及如何使用 Rave 组件制作报表及如何制作安装程序等。

本书由龚涛策划,除第13章由杨锋编写以外,全书由蒙祖强和龚涛执笔、尹江霞审阅。此外,参与本书编写、资料整理或调试编程的还有黎陟、陈哲、戴博、尹江霞、杨林锋、陈燕、黄柏雄、夏洁、罗一丹、江中央、刘星宝、熊琴、申丽曼、刘果、齐龙波、张海滨、孙俊、贺阳剑、樊小虎、贾秀玲、廖玲、杨成立、陶继平、陈曦、徐盛、申舒含、陈玉旺等。

本书作者感谢所有关心和支持本书写作与出版的人员,包括中南大学和广西大学的一些老师、研究生和技术人员,以及中国水利水电出版社计算机图书编辑部的领导和编辑。最后,还特别感谢作者的父母和朋友,他们的关心、帮助和支持使本书得以快速与读者见面。

由于编者水平有限,加上时间仓促,书中疏漏和不当之处在所难免,恳请广大读者批评指正。作者 E-mail: mengzuqiang@163.com。

编 者

2006 年 11 月

目 录

前言

第 1 章 学习 Delphi 的理由和 Delphi 程序员的价值.....	1
1.1 学习 Delphi 的理由	1
1.1.1 Delphi 的发展史.....	1
1.1.2 Delphi 程序员的反思.....	2
1.1.3 Delphi 与 Visual C++ 的比较	4
1.2 Delphi 程序员的价值	7
1.3 Delphi 程序员的编程规则	8
1.3.1 窗体是类	8
1.3.2 继承	12
1.4 小结	14
第 2 章 第一个 Delphi 程序实例的练习	15
2.1 Delphi 2005 集成开发环境 (IDE) 的创建	15
2.2 熟悉 Delphi 2005 的集成开发环境	15
2.2.1 熟悉主窗口	16
2.2.2 工具面板	18
2.2.3 代码编辑器	19
2.2.4 窗体设计器	22
2.2.5 对象观察器	24
2.2.6 工程管理器	25
2.2.7 对象树浏览器	26
2.2.8 模型视图	26
2.2.9 帮助系统	27
2.3 开发第一个 Delphi 应用程序	28
2.3.1 创建工程	28
2.3.2 窗体设计	28
2.3.3 代码编写	29
2.3.4 实例运行	30
2.4 小结	31
第 3 章 熟悉 Object Pascal 语言的编程技术	32
3.1 Object Pascal 语言简介	32
3.1.1 什么是 Object Pascal	32

3.1.2	Delphi 文件及其结构分析.....	32
3.2	熟悉 Object Pascal 的语法要素.....	37
3.2.1	Object Pascal 字符集与标识符.....	37
3.2.2	Object Pascal 保留字.....	37
3.2.3	常量与变量.....	38
3.2.4	运算符.....	39
3.2.5	注释符.....	40
3.3	熟悉 Object Pascal 的数据类型.....	40
3.3.1	简单数据类型.....	41
3.3.2	字符串类型.....	44
3.3.3	指针类型.....	46
3.3.4	结构类型.....	47
3.3.5	过程类型.....	50
3.3.6	可变类型.....	50
3.4	熟悉 Object Pascal 的基本运算.....	50
3.4.1	赋值运算.....	50
3.4.2	算术运算.....	51
3.4.3	逻辑运算与关系运算.....	51
3.4.4	字符串运算.....	52
3.4.5	指针运算和地址运算.....	52
3.4.6	位运算.....	53
3.4.7	运算符的优先级.....	55
3.5	熟悉 Object Pascal 的流程控制.....	55
3.5.1	if 语句.....	55
3.5.2	case 语句.....	61
3.5.3	for 语句.....	62
3.5.4	while 语句.....	65
3.5.5	repeat 语句.....	67
3.5.6	转移语句.....	67
3.6	熟悉 Delphi 的过程和函数.....	68
3.6.1	过程的定义与调用.....	68
3.6.2	函数的定义与调用.....	70
3.7	熟悉 Delphi 中的面向对象编程技术.....	70
3.7.1	熟悉对象与类.....	70
3.7.2	熟悉类的声明.....	71
3.7.3	创建对象及对象成员的引用.....	72
3.7.4	熟悉构造函数与析构函数.....	74

3.7.5 熟悉类的封装、继承和多态性.....	75
3.8 小结	79
第4章 Delphi 中消息和事件的处理	80
4.1 Windows 消息	80
4.1.1 消息记录	80
4.1.2 消息的驱动机制	85
4.2 Delphi 中捕获和处理 Windows 消息	86
4.3 Delphi 消息系统	92
4.3.1 VCL 内部消息	92
4.3.2 通知消息	94
4.3.3 自定义消息与消息的发送	94
4.4 VCL 事件系统与消息	100
4.5 VCL 消息处理机制	102
4.6 小结	104
第5章 GUI 应用程序开发	105
5.1 关于 GUI 应用程序	105
5.2 按钮组件	107
5.2.1 Button 组件	107
5.2.2 BitBtn 组件	109
5.2.3 CheckBox 组件与实例	110
5.2.4 RadioButton 组件与实例	116
5.2.5 SpeedButton 组件	119
5.3 文本组件	119
5.3.1 Label 组件	119
5.3.2 Edit 组件	120
5.3.3 Memo 组件与实例	122
5.3.4 MaskEdit 组件	125
5.3.5 SpinEdit 组件	125
5.3.6 StringGrid 组件	126
5.3.7 RichEdit 组件	126
5.4 列表组件	127
5.4.1 ListBox 组件与实例	127
5.4.2 ComboBox 组件	132
5.4.3 TreeView 组件	133
5.4.4 ListView 组件	134
5.5 对话框组件	134
5.5.1 OpenFileDialog 组件	135

5.5.2	SaveDialog 组件	136
5.5.3	FontDialog 组件	136
5.5.4	ColorDialog 组件	136
5.5.5	PrintDialog 组件和 PrintSetupDialog 组件	136
5.5.6	FindDialog 组件和 ReplaceDialog 组件	137
5.5.7	开发实例——对话框的应用	137
5.6	分类组件	140
5.6.1	GroupBox 组件和 RadioGroup 组件	140
5.6.2	Panel 组件	141
5.7	菜单组件	142
5.7.1	MainMenu 组件	142
5.7.2	PopuMenu 组件	142
5.8	工具栏和状态栏组件	143
5.8.1	ToolBar 组件	143
5.8.2	StatusBar 组件	143
5.9	Timer 组件	143
5.10	创建 GUI 应用程序的窗体及实例开发	144
5.10.1	单文档窗体的创建	144
5.10.2	多文档窗体的创建	145
5.11	熟悉 Delphi 窗体的主要属性	149
5.11.1	Action 标签页中的属性	149
5.11.2	Help and Hints 标签和 Layout 标签中的属性	149
5.11.3	Lagecy 标签、Linkage 标签和 Localizable 标签中的属性	150
5.11.4	Miscellaneous 标签中的属性	151
5.11.5	Visual 标签中的属性	152
5.12	创建与设计 Delphi MDI 应用程序的主菜单	153
5.12.1	创建父窗体中的主菜单	153
5.12.2	创建子窗体中的主菜单	155
5.12.3	父窗体和子窗体中菜单的融合	155
5.13	熟悉 Delphi 的菜单事件	157
5.14	开发实例：多文档界面编辑器	158
5.14.1	实现“子窗口管理”菜单项功能	158
5.14.2	实现“文件”菜单项功能	160
5.14.3	实现“编辑”菜单功能	161
5.14.4	执行多文档窗体应用程序	161
5.15	开发实例：具有查找和替换功能的 RTF 编辑器	163
5.15.1	功能设计	164

5.15.2	程序创建与界面设计	164
5.15.3	主要步骤和核心代码的实现.....	165
5.15.4	执行程序	171
5.16	小结	172
第 6 章	熟悉 Delphi 的 SQL 和 SQL Server 2000 的数据库管理.....	173
6.1	了解关系数据库	173
6.2	熟悉 Delphi 的数据库管理工具	174
6.2.1	BDE Administrator	174
6.2.2	Database Explorer.....	175
6.3	熟悉 Microsoft SQL Server 2000.....	176
6.3.1	安装 SQL Server 2000.....	176
6.3.2	创建与管理 SQL Server 数据库.....	180
6.3.3	创建与管理 SQL Server 数据库表.....	183
6.3.4	SQL Server 数据库的查询设计	185
6.4	熟悉 SQL Server 的基本数据类型和开发工具.....	186
6.5	熟悉 SQL 的定义功能.....	187
6.5.1	创建表	187
6.5.2	创建表的索引	188
6.5.3	创建视图	188
6.6	熟悉 SQL 的数据插入功能.....	190
6.7	熟悉 SQL 的查询功能.....	191
6.7.1	基本查询语句	191
6.7.2	带 where 的条件查询.....	192
6.7.3	带 distinct 的查询.....	192
6.7.4	有序查询	193
6.7.5	带 between 的查询	194
6.7.6	带 in 的查询	195
6.7.7	带 group 的查询	196
6.7.8	带 like 的查询和空值 null 的查询——实现模糊查询.....	197
6.8	熟悉 SQL 的嵌套查询.....	198
6.9	熟悉 SQL 语言的更新功能.....	199
6.9.1	数据的更新	199
6.9.2	表结构的更新	200
6.10	熟悉 SQL 的数据删除功能.....	202
6.11	熟悉 SQL 的库函数.....	203
6.11.1	count 函数.....	203
6.11.2	sum 函数、avg 函数、max 和 min 函数及 round 函数.....	204

6.11.3	mod 函数、power 函数、floor 函数和 sign 函数	204
6.12	熟悉 SQL 的多表处理功能	206
6.13	小结	207
第 7 章	基于 BDE 的数据库应用开发	208
7.1	关于 BDE	208
7.2	熟悉数据库的连接和断开	209
7.2.1	第一个 BDE 数据库应用程序	209
7.2.2	常用的 BDE 组件及相关的数据库组件	212
7.2.3	数据库连接的打开和关闭	214
7.3	熟悉数据库的浏览和查询设计	215
7.3.1	Query 组件提供的方法	215
7.3.2	一个数据浏览实例	216
7.3.3	数据查询设计	220
7.4	熟悉数据库的更新设计	221
7.4.1	数据的插入	221
7.4.2	数据的删除	222
7.4.3	数据的修改	223
7.5	Delphi BDE 数据库的应用实例	223
7.5.1	程序功能设计	223
7.5.2	数据库设计	224
7.5.3	创建 ODBC 数据源	224
7.5.4	创建 Delphi 应用程序	225
7.5.5	成绩输入子系统的设计与实现	228
7.5.6	成绩删除子系统的设计与实现	233
7.5.7	生成期评成绩子系统的设计与实现	235
7.5.8	成绩查询统计子系统的设计与实现	239
7.5.9	运行程序	243
7.6	小结	246
第 8 章	基于 ADO.NET 的数据库应用开发	247
8.1	关于 ADO.NET	247
8.1.1	ADO.NET 简介	247
8.1.2	一个 ADO.NET 数据库应用程序	248
8.2	熟悉常用的 ADO.NET 组件	254
8.2.1	SqlConnection 对象	254
8.2.2	Command 对象	255
8.2.3	DataReader 对象	256
8.2.4	一个应用实例	257

8.2.5 DataSet 对象.....	264
8.3 使用 DataGrid 对象浏览数据	267
8.4 ADO.NET 数据库开发实例	272
8.4.1 实例的设计与数据显示	272
8.4.2 使用 ADO.NET 组件添加数据	273
8.4.3 使用 ADO.NET 组件更新数据	275
8.4.4 使用 ADO.NET 组件删除数据	277
8.4.5 完整的代码设计及程序运行	278
8.5 小结	291
第 9 章 基于 dbExpress 的数据库应用开发	293
9.1 了解 dbExpress	293
9.2 建立第一个 dbExpress 数据库应用程序	295
9.3 熟悉常用的 dbExpress 组件	297
9.3.1 TSQLConnection 组件	297
9.3.2 TSQLDataSet 组件	300
9.3.3 TSQLQuery 组件	301
9.3.4 TSQLTable 组件	302
9.4 熟悉 dbExpress 数据库的连接方法	302
9.4.1 使用已有的数据库连接	303
9.4.2 创建新的数据库连接	303
9.5 熟悉 dbExpress 数据库中的数据管理技术	304
9.5.1 使用 dbExpress 数据集组件	304
9.5.2 使用 TSQLConnection 组件	305
9.5.3 使用 ClientDataSet 组件	305
9.6 dbExpress 数据库开发实例	306
9.6.1 界面设计和属性设置	306
9.6.2 代码设计——实现数据添加、更新和删除功能	307
9.6.3 dbExpress 数据库中的查询设计	309
9.7 小结	311
第 10 章 基于 BDP 的数据库应用开发	312
10.1 了解 BDP	312
10.2 熟悉常用的 BDP 组件	313
10.2.1 BdpConnection 组件	313
10.2.2 BdpCommand 组件	314
10.2.3 BdpDataAdapter 组件	315
10.2.4 BdpDataReader 组件	317
10.3 熟悉基于 BDP 的数据浏览技术	317

10.3.1	创建 BDP 对象.....	317
10.3.2	开发 BDP 对象.....	318
10.3.3	浏览数据	320
10.4	BDP 数据库的开发实例.....	320
10.4.1	数据插入模块的设计与实现.....	321
10.4.2	数据更新模块的设计与实现.....	323
10.4.3	数据删除模块的设计与实现.....	331
10.5	小结	333
第 11 章	熟悉 Delphi 中报表设计和数据输出.....	334
11.1	了解 Delphi 的数据库报表	334
11.2	制作 Delphi 的第一个数据库报表.....	335
11.3	熟悉常用的 Rave 组件	338
11.3.1	RvDataSetConnection 组件.....	338
11.3.2	RvProject 组件.....	339
11.3.3	RvSystem 组件	340
11.4	基于 Delphi Rave 组件的动态报表设计	341
11.4.1	使用 RvSystem 组件动态生成报表	341
11.4.2	使用 RvProject 组件动态修改报表内容.....	344
11.4.3	使用 RvNDRWriter 组件输出报表到文件中.....	347
11.4.4	一个动态报表设计实例.....	348
11.5	小结	354
第 12 章	熟悉 Delphi 中 ASP.NET 应用程序开发.....	355
12.1	了解 ASP.NET	355
12.1.1	ASP.NET 的特点.....	355
12.1.2	ASP.NET 结构.....	356
12.2	配置运行环境及创建 ASP.NET 应用程序	357
12.2.1	配置运行环境	357
12.2.2	安装 IIS 5.1.....	357
12.2.3	创建 ASP.NET 应用程序.....	358
12.3	熟悉 ASP.NET 应用程序的文件结构	360
12.3.1	.dll 文件和.aspx 文件	360
12.3.2	Web.config 文件	363
12.3.3	Web 窗体文件.....	366
12.4	熟悉常用的 ASP.NET 组件	368
12.4.1	Button 和 ImageButton 组件.....	368
12.4.2	TextBox 组件.....	370
12.4.3	ListBox 组件.....	370

12.4.4	CheckBox 和 CheckBoxList 组件.....	371
12.4.5	RadioButton 组件、RadioButtonList 组件和 Table 组件.....	371
12.4.6	HyperLink 和 LinkButton 组件.....	376
12.4.7	RegularExpressionValidator、RequiredFieldValidator、 RequiredFieldValidator 和 ValidationSummary 组件.....	377
12.5	熟悉 Web Form 的指令和语法.....	380
12.5.1	Web Form 指令.....	380
12.5.2	Web Form 语法.....	380
12.5.3	ASP.NET 内置对象.....	382
12.6	设计 Web Form.....	383
12.6.1	Web Form 设计环境.....	383
12.6.2	设置页面字体属性.....	384
12.6.3	使用表格.....	385
12.6.4	插入图片.....	386
12.6.5	创建超链接.....	386
12.7	执行页导航及参数传递.....	387
12.7.1	使用 Form 传递（提交）数据.....	387
12.7.2	使用超链接传递数据.....	389
12.7.3	页面重定向和参数传递.....	389
12.7.4	使用 Session 对象传递参数.....	389
12.8	ASP.NET 数据库开发.....	391
12.9	ASP.NET 应用程序开发实例：网上商店.....	394
12.9.1	创建 ASP.NET 应用程序及其界面设计.....	394
12.9.2	程序代码设计.....	395
12.9.3	程序的运行和使用.....	403
12.10	小结.....	404
第 13 章	熟悉 Delphi 中 ASP.NET Web Services 应用程序开发.....	405
13.1	Web 服务（Web Services）简介.....	405
13.1.1	什么是 Web 服务.....	405
13.1.2	Web 服务的优缺点.....	405
13.1.3	Web 服务体系结构.....	406
13.2	熟悉 Web 服务协议栈.....	407
13.2.1	SOAP.....	407
13.2.2	WSDL.....	407
13.2.3	UDDI.....	408
13.3	创建 Web 服务.....	409
13.3.1	创建 Web 服务应用程序的基本步骤.....	409

13.3.2	熟悉 Web 服务程序的文件结构	411
13.4	Delphi 中 Web 服务应用的开发实例	417
13.4.1	创建 Web 服务程序	417
13.4.2	添加 Web 服务的功能	418
13.4.3	发布 Web 服务	419
13.5	Web 服务的访问实例	419
13.5.1	创建 Web 窗体客户端程序	419
13.5.2	创建 Windows 客户端程序	425
13.6	小结	433
第 14 章	Delphi 2005 数据库应用开发实例——学生信息管理系统	434
14.1	实例系统的总体设计	434
14.1.1	学生信息管理系统概述	434
14.1.2	实例系统的功能描述	435
14.2	实例系统的数据库设计	435
14.3	创建实例系统的数据表	438
14.3.1	创建新的数据库	438
14.3.2	创建表	440
14.3.3	创建实例系统的数据库连接	443
14.4	创建实例系统的应用程序	445
14.4.1	主程序设计	445
14.4.2	学生信息管理模块程序设计	449
14.4.3	教师信息管理模块程序设计	456
14.4.4	课程信息管理模块程序设计	458
14.4.5	成绩管理模块程序设计	461
14.4.6	学生个人成绩单打印功能模块设计	464
14.5	实例系统的安装程序制作	469
14.6	小结	472

第1章 学习 Delphi 的理由和 Delphi 程序员的价值

有不少程序员讨厌诸如英语、日语之类的自然语言，除了 BBS 上的“黑客语”以外，他们只对 C、Pascal 之类的程序设计语言感兴趣。就像李白喜欢写诗而李清照喜欢写词一样，程序员们也总有自己最偏爱的一种或两种程序设计语言。正如有的程序员所说的那样，“程序员将开发工具视作兵器，兵器不称手，未战先败一半。”

Delphi 语言是一种深受许多程序员喜爱的程序设计语言。对于 Delphi 程序员来说，Delphi 带来的是自由而不是束缚，能将一切化繁为简。总之，不管程序员现在是否喜欢 Delphi，都有不少理由让程序员学习 Delphi，因为 Delphi 的确有其独特的优势，通过 Delphi 语言程序员能实现自己的特定价值。简单地说，Delphi 简单、易学，适合于快速开发。

1.1 学习 Delphi 的理由

既然 Delphi 等开发工具是诸位程序员的“兵器”，了解和熟悉兵器是武将的首要任务，因此，有必要首先深入了解什么是 Delphi。

1.1.1 Delphi 的发展史

Delphi 是 Borland 公司（Inprise 公司的前身）推出的一种可视化、方便快捷的 Windows 应用程序开发工具。自从 Delphi 1.0 推出以来，已相继出现了 Delphi 3.0、5.0、6.0、7.0 及 8.0 版本，于 2004 年 11 月推出的 Delphi 2005 是 Delphi 的最新版本。Delphi 的基础语言是 Object Pascal，这种语言是一种强类型语言，相比之下，它采用了优化的编译模式，提供了一种快速的编译器，有效地提高了代码的质量和编译速度。这使得 Delphi 可以与 Visual Basic、Visual C++ 等齐名。

如果说 Delphi 1.0~7.0 版本的更新是 Borland 公司正常的版本升级，那么从 7.0 版本升级到 8.0 版本乃至当今的 Delphi 2005 则是 Borland 公司应对微软的 .NET 技术而作出的英名决策，迎合了软件开发的主流方向。Borland 公司于 2003 年 12 月正式宣布推出支持 Microsoft .NET 框架的 Delphi 8.0 版本（Delphi 8）产品。这一版本较以前版本的改进之处在于提供了对 .NET 的技术支持，可以有效地帮助 Delphi 开发人员实现从现有的 Win32 Delphi 向 .NET 框架的过渡，从而实现基于 .NET 框架的 Delphi 应用程序开发。

Delphi 8 的主要特点有以下几个方面：

(1) Delphi 8 对 .NET 的兼容，可有效改善程序的操作性、安全性和可靠性。Delphi 8 提供基于标准 .NET 的开发环境，完全支持像 ASP.NET、ADO.NET 等基本的组件和标准，从而可利用 .NET 的技术优点，提高 Delphi 产品的安全性、可靠性和可操作性。

(2) 使用 Delphi 8 可以有效地缩短产品的开发周期，集 .NET 技术优点于一身，提高 Delphi 产品的开发成本。 .NET 框架提供了大量的组件，使得许多复杂的问题通过对这些组件的运用而变得非常简单。

(3) 使用 Delphi 8 可开发高性能的 Web 解决方案。Delphi 8 提供了 LiveTools 可视化开发环境和动态集成的 HTML 编辑器, 为高效使用 ASP.NET 组件提供了条件, 从而有助于更快地提供功能强大的电子商务解决方案。

(4) 使用 Delphi 8 可方便开发基于 Web 数据库的产品。用于 ADO.NET 的数据提供者 (BDP) 提供了对 Microsoft SQL Server、Borland InterBase 与 DB2 的支持, 使得 Delphi 程序员能更方便、更灵活和更快捷地开发出满足各种需要的企业级 Web 数据库产品。

虽然, Borland 公司宣称“Delphi 8 是纯正的 .NET, 也是纯正的 Delphi”, 但在实际运用当中还存在不少不尽如人意的地方, 以至于很多人还是停留在 Delphi 6、7 当中, 更有人把它称之, 即“半成品”。也许正是出于对这方面的考虑, Borland 公司于 2004 年 11 月又推出了当今最新的 Delphi 版本, 即 Delphi 2005。

Delphi 2005 的改进功能主要包括以下几个部分:

(1) 支持多种编程语言及 Windows SDK。Delphi 2005 支持现有 Windows 开发所需的编程语言及 SDK, 它同时支持 Delphi 及 C#, 是市场上惟一能真正以单一工具及单一编程语言支持原生 (native) Win32 及 .NET 的开发环境, 并支持 ASP.NET、ADO.NET、VCL.NET 及 Win32 的 VCL。

(2) 结合 ALM 解决方案。Delphi 2005 整合了 StarTeam 及 Optimizeit 的功能, 使开发人员得以检视软件开发周期的各个阶段。StarTeam 组件能简化源代码管理及改进团队沟通, 而内建的 Optimizeit Profile for .NET 则有助于把组件测试自动化, 并改善软件的整体素质与效能。

(3) 实现快速企业模型驱动架构 (MDA) 开发。Delphi 2005 的 ECO II 技术为 .NET 提供了企业级快速 MDA 解决方案, 可加快开发进程、改善成品素质, 并使复杂软件变得更易管理。ECO II 能自动绘制模型图表及创制对象, 从而建立高度可扩展的 .NET 对象群组, 有关对象还具有先进的企业对象功能, 如还原/再做、持久性、版本控制及交易。

(4) 简化及加速 Windows 开发。Delphi 2005 具有一系列创新的 IDE 功能, 能改善开发人员的日常工作体验、增进生产力及精简源代码维护。这些功能包括先进的源代码重构 (Refactoring)、帮助视图 (Help Insights) 及错误视图 (Error Insights) 等深入浅出的系统说明, 以及 SyncEdit、历史管理和新增的 Delphi 功能。RAD for ADO.NET 功能可加速及简化以 Delphi 或 C# 开发连接数据库的 .NET 应用。

Delphi 2005 使用时间还不长, 其优、缺点值得开发人员以后去评说, 但作为基于 .NET 框架的软件开发工具, Delphi 2005 肯定是 Delphi 程序员首选的, 因为基于 .NET 技术的软件开发方法肯定是今后软件开发的主流。

Delphi 2005 有三个版本, 即 Borland Delphi Architect、Borland Delphi Enterprise 和 Borland Delphi Professional。Borland Delphi Architect 适合用于模型驱动应用开发, Borland Delphi Enterprise 常用于建立企业级数据库应用开发, 而 Borland Delphi Professional 则用于个人建立网络和图像化应用开发。作为学习用, 本书主要以 Borland Delphi Architect 为例, 展开对基于 Delphi 2005 的软件应用开发进行阐述。由此, 读者可以方便地引伸到其他版本的学习和应用开发当中。

1.1.2 Delphi 程序员的反思

简单性是这个世界上最难获得的东西, 它是经验的最终界限, 也是天才的最终努力目标。

熟练的 Delphi 程序员可以运用 Delphi 快速地写出一个漂亮、实用的程序，并且热爱 Delphi。Delphi 已经成了程序员工作、学习中不可或缺的一部分。因此，程序员当初选择 Delphi 作为其首选开发工具一定有自己的理由或者原因。简单性就是其中最明显的理由之一。

不少程序员在读高中的时就开始接触计算机。首次学习的编程语言也许是 Basic，那时懂得了程序是由顺序、分支、循环三种结构组成的。然后发现 Delphi 后就转到了 Borland Delphi。也许第一个接触的 Delphi 版本是 3.0 的。促使程序员抛弃其他编程工具而投入 Delphi 怀抱的最初理由简单到可以用一句话加以概括，那就是“Delphi 可以静态连接编译使可执行文件得以独立运行”。这个特点对于发布绿色共享软件来说很重要。

但是，随着 Delphi 版本的升级，人们对 Delphi 的认识也越来越深。它是有着丰富内涵的工具，程序员越了解它，就越喜欢它，也越感觉离不开它，虽然它也还是工具。Pascal 是一种讲究程序美学的语言，基于 Object Pascal（一种支持面向对象的 Pascal 语言）的 Delphi 显得更加完美。

1. 编程语言的相通性和 Delphi 的特色

千变万化的程序归根结底是由顺序、循环、分支三种结构构成；无论是 VC 的 MFC，还是 Delphi 的 VCL，都是由面向对象技术构建的。当程序员拨开事物表面的表象后，看到的就是相同的或近似的本质。而掌握了本质之后，就会发现表象的表现形式是那么的理所当然。从这个角度来说，对于一名专业程序员，编程的理念是万变不离其宗的。发现问题、分析问题、解决问题的过程是存在着某种规律的，当程序员掌握了这种规律后，不同的编程语言、不同的开发环境对程序员来说，是有共同之处的。

首先，通过 C++ 学习面向对象的设计、编程方法是十分必要的。在 C++ 中，程序员可以学习到面向对象理论的全部，学习之后，程序员会有所顿悟。因为在面向对象理论中存在的、却有所争议的特性，比如多重继承，在 C++ 中都得到了支持。只有在掌握这些之后，程序员才可能作出自己的选择。在掌握了面向对象的理论之后，无论 C++、Object Pascal 或是 Java 乃至 C#，程序员都会洞悉到其异曲同工之处。

那是否就是说开发工具之间除了支持的语言不同之外，不存在其他差异了？当然也不是。开发工具是帮助程序员实现其编程设计的工具，也就是构建在基础理念上的上层建筑。开发工具对程序员所要实现的编程设计的支持程度以及对实现过程的简化程度，就是开发工具的友好度了。开发工具对于程序员来说，犹如兵器对于士兵，兵器不顺手，未战先败一半。

在 Windows 平台上，目前有许许多多的开发工具可以选择，如 Visual C++、Visual Basic、Delphi、C++ Builder、Jbuilder、…，它们基于不同的编程语言、忠于不同的公司的产品理念，从这个角度来说，它们之间的差异是非常大的。

那什么样的开发工具才是优秀的、友好的、易用的？Delphi 就是其中的一个典型工具。Delphi 将一切问题化繁为简，却从不阻止程序员寻求真知。程序员可以在 Delphi 所构造的简化了的 VCL 虚拟世界中完成任务，也可以钻进 VCL 的世界以探询 Delphi 和现实世界的映射关系，学习它的代码框架的设计。程序员还可以扩展虚拟的 VCL 世界以适应自己的需要。所以，有的程序员说：“真正的程序员用 C++，聪明的程序员用 Delphi。”

2. Delphi 的优势

对于这么多的主流开发工具，为什么还是选择了 Delphi？也许是没有能够深入地熟悉其他开发工具而导致的，但 Delphi 本身的优秀至少是原因之一。Delphi 的优势主要有以下几点：

(1) 开发高效。Delphi 是一个 RAD (Rapid Application Development, 快速开发工具), 它有可视化的开发环境, 并且 Delphi 有其独到之处。Delphi 是真正面向对象的。其基于 OO 技术构建的 VCL 库中的所有组件都可以被继承以创建新的组件, 包括窗体类 TForm。相比之下, ActiveX 组件缺乏这种灵活性。Delphi 的 CodeInsight 技术 (即代码自动完成功能) 是建立在编译器信息上的, 而 VB 使用的是类型库信息, 使用编译器信息的好处是更具灵活性。

(2) 语言优美。Delphi 是基于 Object Pascal 的语言, 是一种真正支持面向对象而又优雅美观的语言。其在功能的健全上毫不逊色于各种其他面向对象的语言, 但同时又不贪多, 盲目地增加复杂性。使得开发者运用各种模式进行设计时都能得到完善的支持, 实现时却不用考虑太多语言/编译器细节。

(3) 编译高效。Delphi 可以称得上是 Windows 平台上最快的高级语言本地代码编译器了。编译速度的迅捷带来了诸多好处, 其中之一就是快速的编译器可以让程序员频繁地在修改代码和编译运行的状态间切换。

(4) 执行高效。Delphi 不但编译速度快, 生成的目标代码的执行效率也非常高。Delphi 与 C++Builder 使用的是同一个后端优化器, 因此其生成的代码的效率与优秀的 C++编译器生成的代码相同。

Delphi 生成完全本地代码, 因此 Delphi 编译结果的可执行文件可以独立执行、分发, 不需要其他运行库的支持。当然, 程序员也可以选择动态链接编译, 这样可以大大减小可执行文件的长度, 不过这种情况下在分发程序时, 必须同时分发必要的运行库文件。

(5) 维护简便。Delphi 程序员会在一定程度上被限制在 VCL 提供的框架中, 相对来说, 比其他开发工具更容易建立良好设计的代码。代码框架的优良使得软件维护成本大大降低。

1.1.3 Delphi 与 Visual C++ 的比较

下面以一个程序员的角度, 从技术水平、功能、性能、易用性、稳定性、发展历程和前景等方面, 较客观地比较 Visual C++ 和 Delphi 这两大主流开发工具的优、缺点, 其中将涉及语言、应用框架、控件、编译和连接、集成界面、调试、COM、数据库开发等。值得一提的是, 由于 C++Builder 与 Delphi 同为 Inprise 公司的产品, 它们除了使用的语言不同, 其余特性几乎完全相同。

1. 编程语言的比较

VC 和 Delphi 本身不是语言, 而是开发平台。它们所用的语言分别是略作扩展的 C/C++ 和 Object Pascal。下面比较一下 C++ 和 Object Pascal 的优、缺点。

有人认为 Object Pascal 是“玩具语言”, C++ 才是“专业语言”, 这是不对的。单从语言本身看, Object Pascal 与 C++ 属同一重量级。它们都是完全支持面向对象的语言, 都是扎根于面向过程的语言。C++ 是由 C 发展而来的, Object Pascal 由 Pascal 进化而来。它们都有很强的灵活性, 都有自己的特长和不足。比如说, Object Pascal 不支持多重继承、模板、操作符重载、内联函数定义、预处理、宏、全局静态类变量、嵌套类定义等, 而这些都是 C++ 支持的。但同样地, C++ 也不支持 Object Pascal 的虚构造函数、过程嵌套、内置集合类型、内置字符串类型、finally 构造等, 在 RTTI 方面 Object Pascal 也比 C++ 做得好。但这些并不重要, 因为可以通过其他方式达到同样的目的, 比如 C++ 可以通过类扩展支持集合、字符串, Object Pascal 可以通过 interface 实现多重继承等。关键是二者都可以很好地完成程序员手头的任务。

但是, 仅仅比较语言本身还是不够的, 还得看它们的被接受和流行程度、学习曲线、发展前途、可移植性等, 以及很重要但常常被忽略的一点: 与开发环境及其应用框架的“磨合”程度。

VC 和 Delphi 作为开发平台都提供了应用框架, 即 VC 的 MFC 和 Delphi 的 VCL。MFC 是用 C++ 编写的, VCL 是用 Object Pascal 编写的。当然, C++ 的使用范围比 Object Pascal 广得多, 移植性也好得多。这本来是优点, 但正因为如此, 微软在编写 MFC 时不得不考虑最大限度地减少对语言本身的改动, 而把功夫下在源代码级, 以便尽可能支持 ANSI 等标准, 结果导致 MFC 的封装复杂而不直观。太多的宏定义和含义模糊且自动生成、不得改动的注释使 MFC 乃至 VC 让很多新手望而生畏, 不敢深入学习。而 Object Pascal 几乎是 Inprise 专用的, 不必考虑“标准”问题, 因此, 采用 Inprise 编写 VCL 时就把全部精力放在了结构与性能上, 结果语言与框架的磨合程度非常好。VCL 框架的结构清晰, 代码的可读性非常好。许多人说 Delphi 比较容易上手, 也是这个缘故。

2. 编译和连接的比较

不同的语言导致了编译和连接的速度不同, 以及执行程序的速度不同。Delphi 的编译和连接速度, 毫不夸张地说, 比 VC 快几十倍。即使把 VC 的 Incremental Link 选项打开, Delphi 的编译和连接速度仍比 VC 快好几倍。并不是说微软的编译器不行, 这是由 C++ 的复杂性决定的。模板的处理、预处理和宏的展开都是很费时的。这本来是缺点, 但带来的一个好处就是编译速度极快。至于编译完的二进制代码, 在打开相同的优化选项的情况下, Delphi 和 VC 执行速度并没有太大的差别。

两个编译器有一个共同点就是都能识别无用的代码, 比如一个没有用的函数等。编译后的程序将不包含这些多余的信息。Delphi 在这方面做得更加出色, 它可以让程序员在编辑器中可视化地提示出哪行代码是要用到的、哪行代码是没有用到的。这样程序员就能整理出最精简的代码。Delphi 在编译后将在左边显示一个小蓝点表示这行代码是用到的, 而 Visual C++ 做不到这一点。

Delphi 编译后可执行文件至少有 200K (如果不使用 VCL, 仅使用 WinAPI, 文件的大小将大大缩小), 但是 Visual C++ 编程使用 MFC 编译后的可执行文件通常只有几十 K, 主要是因为微软已经将系统运行库包含在 Windows 系统了。同样, 使用 BDE 开发的数据库程序必须附带 3~5M 的额外系统文件, 也是非常不协调的。Delphi 能够使用由 C++ Builder 创建的 OBJ 文件, 但是使用上受到很大的局限性。

最后, Visual C++ 的编译和连接时的错误信息比 Delphi 要详细和具体得多, 特别是使用 ATL 开发更是如此。

3. 应用框架 MFC 和 KFC 的比较

应用程序框架 (Application Frame) 有时也称为对象框架。Visual C++ 采用的框架是 MFC, MFC 不仅是人们通常理解的类库。同样, Delphi 的 VCL 也不仅是一个控件库, 尽管它的名字叫可视控件库。

最能体现一个应用程序框架先进性的是它的委托模型, 即对 Windows 消息的封装机制。对 Windows API 的封装非常简单, 没有什么技术含量, 在操作上也大同小异, 程序开发人员也可以自己编写一个类库来封装。但对 Windows 消息驱动机制的封装就不是那么容易的了。最自然的封装方式是采用虚成员函数。如果要响应某个消息就重载相应的虚函数。MFC 采用

的是宏定义方法，用宏定义方法的好处是省去了虚函数 VTable 的系统开销，但由于 Windows 的消息种类很多，开销不算太小，不过带来的缺点就是映射不太直观。好在 VC 的 ClassWizard 可以自动生成消息映射代码，使用起来还算方便。但与 VCL 的委托模型相比，MFC 的映射方法就显得太落后了。而 Delphi 的 Object Pascal 因为没有标准负担，语言引入了组件、事件处理、属性等新特性。由于功夫做在编译器级，生成的源代码就显得十分简洁。似乎 VC 是“让框架迁就语言”，而 Delphi 是“让语言迁就框架”。

VCL 与 MFC 相比的另一个优点是对异常处理的支持，而一大缺点是对多线程支持差。VCL 的大部分都不是针对多线程优化的。虽说 VCL 提供了简化多线程操作的类，但只是工作者线程使用起来比较简单。如果线程要和界面打交道的話，事情就变得很复杂了，因为除了应用程序的主线程，任何线程不能访问任何可视的 VCL 部件。程序员不得不使用 Synchronize 方法等待主线程处理它的消息，然后在主线程中访问 VCL 部件。而 MFC 就没有这样的限制。

4. 稳定性与完善程度的比较

VC 具有更悠久的历史，同时 VC 的性能比 Delphi 更趋稳定和完善。VC 的发展历史比 Delphi 长，这得力于微软雄厚的总体实力，它在开发 VC 上比 Inprise 具有更强的技术。VC 的框架 MFC 经历了那么多年的发展和完善，功能非常全面，而且十分稳定，bug 很少，而且有第三方的专门工具帮助程序员避开这些 bug。

5. 可移植性

立足现实，Inprise 已开发出 Delphi 的 Linux 版本，代号为 Kylix。通过 Kylix，用 VCL 构架编写的 Windows 程序向 Linux 移植成为可能，而 MFC1.0 的程序可以毫无障碍地在 VC6.0 下编译通过。

6. 集成界面的比较

从总体上来说，VC 的集成界面是不如 Delphi 的。Delphi 仅具有的一个 Object Inspector 功能就使 VC 的许多 Wizards 功能逊色，何况它还具有 Code Explorer、ToDo List 等功能。但从细微上来说，Delphi 也有一些不成熟的地方。比如，自动完成功能的智能化程度和提示详细程度不如 VC，响应速度也没有 VC 快。

Visual C++ 所带的 MSDN 是一部开发者的百科全书，信息量庞大，查询方便，这方面比 Delphi 更专业。很多帮助项都有源程序示范。

Delphi 的 OpenTools 是完全面向第三方的开放系统，开发者可以修改很多 Borland 公司自身的功能，从 IDE 的可扩充性上说 Delphi 更好。

7. 调试的比较

Visual C++ 和 Delphi 的调试功能都非常强大，同时都具有单步可视化调试、断点跟踪、运行时改变变量、鼠标指向可以得到变量值等功能。对 DLL 的输入、输出也能方便地进行，能够进行源代码级别的调试。

相对而言，Visual C++ 能够更加方便地看到变量的变化情况，这包括对结构可以展开成数据树，从而了解每一个变量的值，每一步调试，变化了的变量会变红，从而使调试更加方便。另外，Visual C++ 的块内存察看比 Delphi 也要方便。

当然，Delphi 也有很多体贴的细微之处，比如在线程调试的时候，Delphi 能够很方便地察看线程的变化，Visual C++ 却必须要弹出一个模式对话框。

8. 数据库开发的比较

数据库支持是 Delphi 的强项,这主要体现在 Delphi 与 BDE 的无缝集成,以及 Delphi 提供的那一大堆现成的数据库操作控件。这是 VC 望尘莫及的。Delphi 支持 BDE、ADO、InterBase 三种数据库访问方式。所有的方式都能拖拉到应用程序中实现可视化操作。正是因为 Delphi 对数据库类的包装,使得用户操作数据库不像在 Visual C++ 中必须从开始到最后都要干预,从而明显地提高了开发速度。

Delphi 中使用 WebBroker 控件,能很方便地构造出基于数据库的 Web 页面,通过 HTML 管理 Web 数据库。Visual C++ 访问数据库主要通过 ADO 和 OLEDB,很多 ActiveX 控件也能添加数据库功能。但是没有像 Paradox 这样的桌面数据库,Access 相对功能太弱了。也许 SQL Server 是不错的选择。

昨天、今天、明天,技术的进步在很多时候是此消彼长的。当初 Borland 公司的 Turbo C 和 Borland C++ 几乎是 C/C++ 程序员惟一的选择。微软的 Quick C 和 Microsoft C/C++ 从来也没有成为过主流。Borland C++ 也被崛起的 Microsoft Visual C/C++ 压下去了。

于是 Inprise(原 Borland)拣起了当年 Turbo Pascal 和 Borland Pascal 的辉煌(事实上 Borland 的成名作就是第一个 Pascal 编译器),全力推出了 Delphi。

Delphi 当初推出时被称为 VB 杀手,但 VB 现在仍然活得挺好。毕竟微软是靠 Basic 起家的嘛,VB 不是那么容易被打败的。Inprise 想了想不和 VB 争了,使用 Delphi 的 IDE 和 VCL 配上 C++ 语言,推出了 C++Builder,又向 Visual C++ 的市场发起了夹攻。C++Builder 似乎是个不错的折衷选择了?再仔细想想!C++Builder 的优点 Delphi 都有,但 Delphi 的优点 C++Builder 却未必有。比如 C++Builder 的编译速度比 VC 还慢,哪能和 Delphi 相比?而且因为 VCL 是 Object Pascal 编写的,C++ 语言和 VCL 磨合得并不好。C++Builder 的 bug 比 Delphi 还多,甚至 Sample 代码中还有错。VCL 的部分功能不能使用,要靠嵌入 Pascal 代码访问。C++Builder 可用的第三方控件远没有 Delphi 的多。

1.2 Delphi 程序员的价值

Delphi 2005 是目前市场上最全面的 Windows IDE(综合开发环境)及 ALM 开发系统之一,也是近年来 Delphi 平台产品中最重要版本。它能帮助开发团队了解软件项目情况并进行进度预测,从而在预算范围准时推出软件,并能最大限度地发挥其业务价值。

Delphi 2005 支持多种编程语言和 Win32 及 .NET 的 SDK(软件开发套件),又具备一系列能提升程序员及团队生产力的功能,包括源代码重构(Code Refactoring)、组件测试及崭新的第二代企业核心对象技术(ECO II)。ECO II 为模型驱动的 .NET 企业应用提供了开发框架。Delphi 2005 还能协助程序员团队在创建新应用的同时,维护及改进现有的 Windows 应用。

Borland 开发工具产品管理总监 Michael Swindell 称:“Borland 非常了解现今 Windows 开发团队所承受的压力:他们既要支持现有应用,又要开始采纳新科技,但开发周期及资源却在缩减。Delphi 2005 提供了开发者所需要的功能,协助加快现有的和下一代 Windows 应用的开发及维护流程。”

Delphi 2005 支持多种 Windows 编程语言、Win32 及 .NET 软件开发套件(SDK)、集成 ALM 功能及程序员生产力促进功能。凭借 Delphi 2005,程序员既能继续改进 Win32 应用,又能先

行采用.NET 及 ASP.NET。

美国亚利桑那州的 Succeed 公司采用 Delphi 为中、小型企业开发管理产品及服务已有十年。该公司只用 Delphi, 创建出 AuctionBlast 系统, 帮助卖家在 eBay 网上市场进行市场拓展。该系统现已是 Succeed 最受欢迎的产品。

Succeed 的行政总裁 Omar Sayed 表示: “Delphi 为我们在软件开发方面带来了难以置信的重大成功。我们相信 Delphi 2005 的新功能将协助我们加速开发, 发挥我们现有投资及技术的潜力, 并把握 eBay 网络服务平台等不断涌现的商机。”

Gartner 研究部副总裁 Mark Driver 表示: “开发团队需要既能方便他们工作, 又能灵活地适应不断转变的业务要求的工具与平台。只要能加快开发速度、改善沟通及生产力和强对整个应用软件开发周期的控制, 开发团队不论大小均能大大受惠。”

总之, Delphi 程序员的价值随着 Delphi 版本的升级也得到提升。Delphi 简单、易用、适用于快速开发的特点让 Delphi 程序员具备了特有的价值。

1.3 Delphi 程序员的编程规则

下面介绍 Delphi 程序员编程的一些经验规则, 制定这些规则的目的在于帮助改善程序员的程序结构。程序员应当把它们当作一些建议, 记住并将其应用到要开发的程序中去。

面向对象编程的一个关键就是封装。经常要创建一些灵活而且强健的类, 因为这样的类允许程序员以后修改其实现方法而不影响到程序中的其他部分, 这正是封装给程序员带来的好处。虽然封装不是创建一个好的面向对象程序的惟一标准, 但是它构成了面向对象编程的基础, 所以这里特别强调封装性。

这里主要以窗体 (Forms) 的开发为例进行的讲述, 因此这些规则对于所有的 Delphi 程序员都是适用的。那些编写组件的程序员必须把面向对象编程和类 (Class) 作为核心的元素, 但是对于那些使用组件编程的程序员, 他们时常会忘记面向对象。

1.3.1 窗体是类

程序员常常将窗体看做对象, 而事实上窗体是类。两者的差别在于程序员创建基于相同的窗体类的多个窗体对象。Delphi 为程序员定义的每一个窗体类创建了一个默认的全局对象。这对于新手来说是相当方便的, 但是这同样会使他们形成坏习惯。

1. 为每一个类创建一个单元

类的私有和保护的部分仅仅对于其他单元中的类和过程才是隐藏的, 因此, 如果程序员想得到有效的封装性, 就应该对每一个类使用一个不同的单元。对于一些简单的类, 比如那些继承其他类的类, 程序员可以使用一个共享的单元。不过共享同一个单元的类的数目是受到限制的, 如不要在一个简单的单元里放置超过 20 个复杂的类。

如果程序员使用窗体的时候, Delphi 会默认地遵循“一个类使用一个单元”的规则, 这对于程序员来说也是十分方便的。当程序员向项目中添加一个没有窗体的类时, Delphi 也会创建一个新的独立的单元。

2. 为组件命名

为每一个窗体和单元给出一个有意义的名字是十分重要的。窗体和单元的名字必须是不

同的, 不过这里趋向于为它们两者使用相似的名字, 如关于窗体和单元可以为它们使用 AboutForm 和 About.pas。

为组件使用带有描述性的名字同样十分重要。最常见的命名方式是使用类的小写字母开头, 再加上组件的功能, 如 BtnAdd 或者 editName。采用这样的命名方式为组件命名可能会有很多相似的名字, 而且也没有一个最好的名字, 到底应该选择哪一个这要依据个人的爱好而定。

3. 为事件命名

对于事件处理方法给出合适的名字更加重要。如果给组件一个合适的名字, 那么系统默认的名字 ButtonClick 将变成 BtnAddClick。虽然从这个名字中可以猜到这个事件处理程序的功能, 但是使用一个能够描述该方法的作用的名字, 而不是采用 Delphi 附加的名字是一种更好的方式。例如, BtnAdd 按钮的 OnClick 事件可以命名成 AddToList。这会使得程序可读性更强, 特别是当程序员在这个类的其他方法中调用这个事件处理程序时, 而且这会帮助程序员为类似的事件或是不同的组件选用相同的方法。

4. 使用窗体方法

窗体都是一些类, 因此窗体的代码是按照方法组织的。程序员可以向窗体中添加事件处理程序, 这些处理程序完成一些特别的功能, 而且它们能被其他方法调用。除了事件处理方法外, 程序员还可以向窗体添加完成动作的特别定义的方法及访问窗体状态的方法。在窗体中添加一些公共的 (Public) 方法, 让其他窗体调用要比其他窗体直接操作其组件要好。

5. 添加窗体构造器

在运行时创建的第二个窗体除了一个默认的构造器 (从 Tcomponent 类继承而来) 外, 还会提供其他特殊的构造器。如果程序员不需要考虑和 Delphi 4 以前版本的兼容性问题, 就重载 Create 方法, 添加必要的初始化参数。具体可参见下面的代码。

```
Public
    Constructor Create(Text:string): reintroduce ; overload;
    Constructor TFormDialog.Create(Text:string);
Begin
    Inherited Create(Application);
    Edit1.Text:=Text;
End;
```

6. 避免全局变量

应该避免使用全局变量 (就是那些在单元的 interface 部分定义的变量)。如果程序员要为窗体存储额外的数据, 就可以向窗体类中添加一些私有数据。在这种情况下, 每一个窗体实例都会有自己的数据副本。程序员可以使用单元变量 (在单元的 implementation 部分定义的变量) 声明那些供窗体类的多个实例共享的数据。

如果程序员要在不同类型的窗体之间共享数据, 就可以把它们定义在主窗体里来实现共享, 或者使用一个全局变量, 使用方法或是属性来获得数据。

7. 永远不要在 TForm1 类中使用 Form1

程序员应该避免在一个类的方法中使用其特定的对象名称, 换句话说, 程序员不应该在 TForm1 类的方法中直接使用 Form1。如果程序员确实要使用当前的对象, 就可以使用 Self 关键字。大多数时候, 程序员没有必要直接使用当前对象的方法和数据。

如果程序员不遵循这条规则, 当程序员为一个窗体类创建多个实例的时候, 就会陷入麻

烦当中。

8. 尽量避免在其他的窗体中使用 Form1

即使在其他窗体的代码中,程序员也应该尽量避免直接使用全局变量,如 Form1.定义一些局部变量或者私有域供其他窗体使用会比直接调用全局变量要好。

例如,程序的主窗体能够为对话框定义一个私有域。很显然,如果程序员计划为一个派生窗体创建多个实例,这条规则将十分有用。程序员可以在主窗体的代码范围内保持一份清单,也可以更简单地使用全局 Screen 对象的窗体数组。

9. 移除 Form1

在程序中移除 Delphi 自动创建的全局窗体对象,即使程序员禁止了窗体的自动添加功能,这也有可能是必要的,因为 Delphi 在随后仍然可能添加这样的窗体。应该尽量避免使用全局窗体对象。

对于 Delphi 新手而言,移除全局窗体对象是十分有用的,这样他们不至于因为类和全局对象两者之间的关系感到疑惑。事实上,在全局窗体对象被移除后,所有与它有关的代码都会产生错误。

10. 添加窗体属性

当程序员要为窗体添加数据时,就添加一个私有域。如果程序员要访问其他类的数据,可以为窗体添加属性。使用这种方法,程序员就能够改变当前窗体的代码和数据(包含在它的用户界面中),而不必改变其他窗体或者类的代码。

程序员还应该使用属性或是方法来初始化派生窗体或对话框,或是访问它们的最终状态。应该使用构造器来完成初始化工作。

11. 显示组件属性

当程序员要访问其他窗体的状态时,就不应该直接访问它的组件。因为这样会将其他窗体或其他类的代码和用户界面结合在一起,而用户界面往往是一个应用程序中最容易发生变化的部分。最好的方法是,为程序员要访问的组件属性定义一个窗体属性。要实现这一点,可以通过读取组件状态的 Get 方法和设置组件状态的 Set 方法实现。

假如程序员现在需要改变用户界面,就用另外一个组件替换现有的组件,那么程序员只需修改与这个组件属性相关的 Get 方法和 Set 方法,而不必查找、修改所有引用这个组件的窗体和类的源代码。详细实现方法可以参见下面的代码。

```
private
    function GetText:String;
    procedure SetText(const Value:String);
public
    property Text:String;
    read GetText write SetText;
end;
function TFormDialog.GetText:String;
begin
    Result:=Edit1.Text;
end;
procedure TFormDialog.SetText(const Value:String);
begin
```

```
Edit1.Text:=Value;  
end;
```

12. 属性数组

如果程序员要处理窗体中的一系列变量, 就可以定义一个属性数组。如果这些变量是一些对于窗体很重要的信息, 就可以把它们定义成窗体默认的属性数组, 这样程序员就可以直接使用 SpecialForm[3]来访问它们的值了。

下面的代码显示了如何将一个 listbox 组件的项目定义成窗体默认的属性数组。

```
type  
TformDialog =class(TForm)  
private  
    listItems:TListBox;  
    function GetItems(Index:Integer):String;  
    procedure SetItems(Index:Integer;const Value:String);  
public  
    property Items[Index:Integer]:string;  
end;  
function TformDialog.GetItems(Index:Integer):string;  
begin  
    if Index >=ListItems.Items.Count then  
        raise Exception.Create('TformDialog:Out of Range');  
    Result:=ListItems.Items[Index];  
end;  
procedure TformDialog.SetItems(Index:Integer;const alue:string);  
begin  
    if Index >=ListItems.Items.Count then  
        raise Exception.Create('TformDialog:Out of Range');  
    ListItems.Items[Index]:=Value;  
end;
```

13. 使用属性的附加作用

使用属性而不是访问全局变量的好处之一就是当程序员设置或者读取属性的值时, 程序员还可能有意想不到的收获。

例如, 程序员可以直接在窗体界面上拖拉组件, 设置多个属性的值, 调用特殊方法, 立即改变多个组件的状态, 或者撤销一个事件 (如果需要的话) 等。

14. 隐藏组件

面向对象编程的狂热追求者常抱怨 Delphi 窗体中包含一些在 published 部分声明的组件, 这是和面向对象思想的封装性原理不相符合的。他们确实提出了一个重要的议题, 但是他们中的大多数人都没有意识到解决方法其实就在他们手边, 完全不用重写 Delphi 代码, 也不用转向其他语言。

Delphi 向窗体中添加的组件参照可以被移到 private 部分, 使得其他窗体不能访问它们。如果程序员这样做, 就有必要设置一些指向组件的窗体属性, 并且使用它们来访问组件的状态。

Delphi 将所有的这些组件都放在 published 部分, 这是因为使用这种方式能够保证这些组件是在.DFM 文件中创建的。当程序员改变一个组件的名称时, VCL 能够自动地将这个组件对象与它在窗体中的组件参照关联起来。因为 Delphi 使用 RTTI 和 TObject 方法来实现这种功能,

所以如果要使用这种自动实现功能,就必须把组件参照放置在 published 部分,这也正是为什么 Delphi 将所有的组件都放在 published 部分的缘故。程序员可以参考下面的代码。

```
procedure Tcomponent.SetReference(Enable:Boolean);
var
  Field:Tcomponent;
begin
  If Fowner<> nil then begin
    Field:=Fowner.FieldAddress(Fname);
    If Field<>nil then
      Field^:=Self
    else
      Field^:=nil;
  end;
end;
```

上面的代码是 Tcomponent 类的 SetReference 方法,这个方法可以被 InsertComponent、RemoveComponent 和 SetName 等方法调用。

当程序员理解了这一点后,就应该不难想到如果将组件参照从 published 部分移到了 private 段,就将失去 VCL 的自动关联功能。为了解决这个问题,程序员可以通过在窗体的 OnCreate 事件中添加如下代码解决:

```
Edit1:=FindComponent('Edit1') as Tedit;
```

接下来,程序员应该做的就是系统中注册这些组件类,当程序员为它们注册过后,就能使 RTTI 包含在编译程序中并能够被系统所使用。当程序员将这些类型的组件参照移到 private 部分时,对于每一个组件类,程序员只需为它们注册一次。即使没必要注册它们,程序员也可以这样做,因为对于 RegisterClasses 的额外调用有益无害。通常,程序员应该在单元中负责生成窗体的初始化部分添加以下的代码:

```
RegisterClasses([TEdit]);
```

15. 面向对象编程的窗体向导

为每一个窗体的每一个组件重复上述两个操作不仅十分麻烦,而且相当浪费时间。为了避免额外的负担,程序员可以为此写一个简单的向导程序。这个程序将会生成一些可以完成以上两步工作的代码,然后程序员要做的工作仅仅是做几次复制和粘贴。

1.3.2 继承

上面讲述了一系列有关类特别是关于窗体类的规则后,下面介绍一些有关类的继承性及可视化窗体继承的建议和技巧。

1. 可视化窗体继承

如果应用得当,这将会是一个强大的工具。根据经验,程序员所开发的项目越大,就越能体现它的价值。在一个复杂的程序中,程序员可以使用窗体的不同等级关系,来处理一组相关窗体的多态性 (polymorphism)。

可视化窗体继承允许程序员共享多个窗体的一些公共的动作,程序员可以使用共享的方法、公用的属性,甚至是事件处理程序、组件、组件属性、组件事件处理方法等。

2. 限制保护域数据的使用

当创建一些具有不同分级体系的类时,一些程序员趋向于主要使用保护域,因为私有数据不能被子类所访问。不能说这没有其合理性,但是这肯定是和封装性不相容的。保护数据的实现能够被所有继承的窗体所共享,而且一旦这些数据的原始定义发生改变,程序员必须更改所有的相关部分。

如果程序员遵循隐藏组件这条规则,继承窗体就不可能访问基类的私有组件。在一个继承窗体中,类似 `Edit1.Text: =` 的代码就不会被编译。虽然这是相当的不方便,但是至少在理论上这是值得肯定的事情。如果程序员意识到实现封装性是最主要和最迫切的,就应该将这些组件参照放在基类的私有段。

3. 保护域中的访问方法

应将组件参照放置在基类的私有域中,而为这些组件添加一些访问函数来得到它们的属性,这将是一种更好的方法。如果这些访问函数仅仅在这些类内部使用而且不是类接口的一部分,程序员应该在保护域声明它们。例如,上一小节中第 11 条规则中描述过的 `GetText` 和 `SetText` 方法,就可以声明成 `protected`, 并且程序员可以通过调用 `SetText ("")` 方法来编辑文本。

事实上,当一个方法被镜像到一个属性时,程序员可以简单地采用如下代码,就可以达到编辑文本的目的。

```
Text:=;
```

4. 保护域中的虚拟方法

实现一个灵活的分级制度的另一个关键点是,定义一些程序员可以从外部类调用的虚拟方法来得到多态性。如果这个方法使用得当,将会很少出现其他公共的方法调用保护域中的虚拟方法的情况。这是一个重要的技巧,因为程序员可以定制派生类的虚拟方法,来修改对象的动作。

5. 用于属性的虚拟方法

即使是访问属性的方法也能定义成 `Virtual`, 这样就能通过派生类改变属性的动作,而不必重新定义它们。虽然这种方法在 VCL 当中很少使用,但是它确实十分灵活、强大。为了实现这一点,仅仅需要将上一小节的第 11 条规则中的 `Get` 和 `Set` 方法定义成 `Virtual`。基类的代码如下。

```
type
TformDialog = class ( TForm)
  Procedure FormCreate(Sender:TObject);
  Private
    Edit1:Tedit;
  Protected
    function GetText:String;virtual;
    procedure SetText(const Value:String);virtual;
  public
    constructor Create(Text :String):reintroduce;overload;
    property Text:String read GetText write SetText;
end;
```

在继承窗体中,程序员可以添加一些额外的动作来重载虚拟方法 `SetText`。

```
procedure TformInherit.SetText(const Value:String);
```



```
begin
    inherited SetText(Value);
    if Value="" then
        Button1.Enabled:=False;
end;
```

1.4 小结

Delphi 发展至今，已成为支持.NET 的强大开发工具，但其最显著的特点在于简单、易学，适合于快速开发。因此，Delphi 程序员也在软件开发世界中有了特定的地位和作用，能实现其特有的价值。

学习 Delphi 有了目的和动力，但要作为一个好的 Delphi 面向对象程序员远非上面提到的规则这么简单。上面的这些规则中有一些可能需要付出足够的耐性和时间，因此，不能强求程序员一定要遵循所有的这些规则。但是这些规则应该被合适地运用到程序中，而且当程序员开发的应用程序越大，参与的程序员越多，这些规则越重要。不过，即使是一些小程序，始终记住这些规则并在合适的地方使用它们也会对程序员有所帮助。

总之，要遵循上面列出的规则就要付出一定的努力，特别是额外的代码，但是这些努力会让程序员得到一个更加灵活、健壮的程序。

第2章 第一个 Delphi 程序实例的练习

万丈高楼平地起，第一个 Delphi 程序实例的练习就是 Delphi 程序员成长之路的第一块也是十分重要的一块基石。在设计并编程实现这个实例之前，先要创建好 Delphi 2005 集成开发环境，然后熟悉 Delphi 2005 的可视化集成开发环境 (IDE)，最后才能编写第一个简单的 Delphi 程序实例。本章的要点如下：

- 了解 Delphi 2005 集成开发环境的基本组成；
- 熟悉 Delphi 2005 的集成开发环境；
- 第一个 Delphi 程序实例的编程步骤。

2.1 Delphi 2005 集成开发环境 (IDE) 的创建

Delphi 2005 的先进和特殊之处在于，它整合了微软公司的 .NET 资源，使得在 Delphi 环境中可以进行基于 .NET 的程序开发。这就需要在安装 Delphi 2005 之前先要安装一些支持性的软件。所以，Delphi 2005 安装过程比较繁琐，与一般应用程序的安装过程不太一样，对系统的要求比较高。在安装 Delphi 2005 之前，必须对下列组件进行安装和升级：

- 安装 IE v6.0 Service pack 2 组件。
- 安装 Microsoft .NET Framework 1.1。
- 安装 Microsoft .NET Framework SDK。
- 安装 Microsoft J# .NET v1.1。

其中，pack 2 组件主要是出于对系统安全的考虑；.NET Framework 1.1 是 Microsoft 提出的一种新的计算机平台，其目的是提供一个简化的高度分布的 Internet 环境中应用程序的开发平台；.NET Framework SDK 是为了更能有效、轻松地创建、布置和管理针对 .NET Framework 的应用程序和组件；Microsoft J# .NET 则是方便在 .NET Framework 这个分布式平台上构建应用程序和服务。Delphi 2005 是支持分布式开发的，所以在其安装前必须安装这些组件。

实际上，现在市场上提供的 Borland Delphi 2005 安装盘都提供这些工具和组件，并且具有能够自动检测这些组件是否已安装的功能。如果没有安装，则安装程序会提示进行安装；如果已经装了，它就不会再作出提示，也不会重新被安装了。因此，读者在安装过程中可能发现，在不同机器上安装过程可能不尽相同，这就不足为奇了。例如，如果是刚刚装完的 Windows XP 系统，在安装 Borland Delphi 2005 时会弹出大量的对话框，需要用户作出选择，安装也会很久，这是很正常的。

总之，在安装过程中要认真阅读对话框的提示信息，并作出正确的选择，就可以成功地完成 Borland Delphi 2005 的安装，构建一个良好的 Delphi 2005 集成开发环境。

2.2 熟悉 Delphi 2005 的集成开发环境

Delphi 2005 具有强大功能的可视化集成开发环境 (IDE)，为用户提供了一种方便、快捷

的应用程序开发方法。用户界面是基于当前最广泛的微软的 Windows 窗口，同时在程序设计上也继承了当前程序设计技术的主流——面向对象的编程技术。因此，不管是在程序设计上还是在界面设计中，基于 Delphi 2005 的软件开发技术将易于为广大程序员所接受，Delphi 2005 是理想、高效的可视化软件开发工具。

集成开发环境主要是由三部分组成，即窗体设计器、调试器和代码编辑器，其中，可视化环境主要体现在窗体设计器上。通过在窗体设计器中对窗体进行创建和构造，设计出满足用户需要的界面；同时，对窗体任何改动都会使得后台的编辑器中代码进行自动更新和生成；另外，在代码编辑器中可以定义相应的代码来控制程序的行为，还可以通过设置断点和监控点等来调试程序。实际上这三部分是协同工作的，它们是一个有机的整体，共同构成了可视化 IDE 的要素。

2.2.1 熟悉主窗口

Delphi 2005 的可视化集成开发环境界面如图 2.1 所示。从图中可以看出，Delphi 2005 的界面主要是由七个部分组成，以下分别对它们进行介绍。本节主要介绍第一部分。

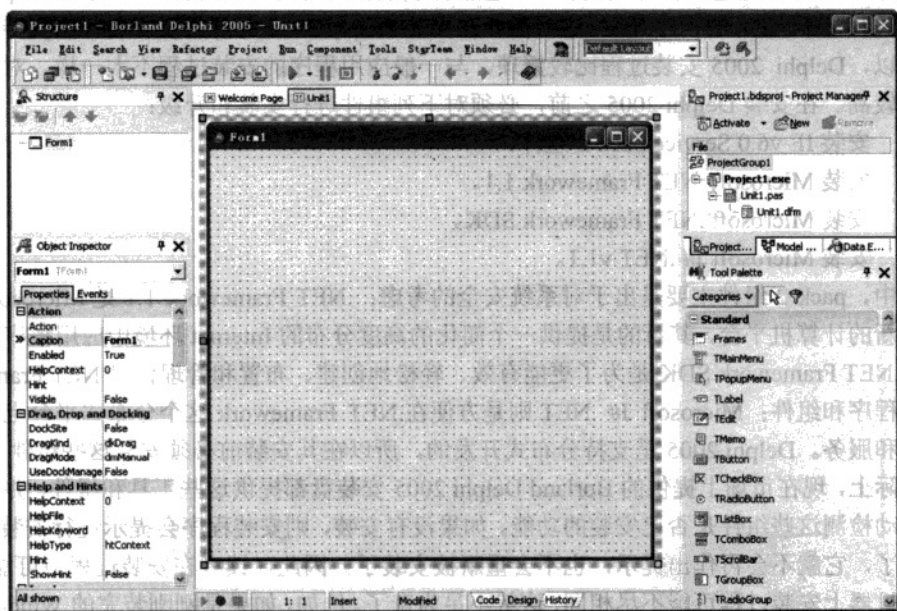


图 2.1 Delphi 2005 的开发环境界面

在默认情况下，主窗口位于界面的最上端，如图 2.2 所示，它包含了 Delphi 2005 的菜单栏、工具栏和标题栏。



图 2.2 Delphi 2005 的主窗口

1. 菜单栏

类似于 Windows 应用程序, Delphi 2005 可以通过菜单栏的命令来完成它特定的功能和任务。其菜单包括 File、Edit、Search、View、Refactor、Project、Run、Component、Tools、StarTeam、Window、Help。菜单中的功能将在后续章节当中陆续学到, 故在此对它们的使用方法不予以说明。

2. 工具栏

工具栏上的按钮实际上是部分常用菜单命令的图形化表示, 即单击工具栏上的某一个按钮也就相当于执行菜单中相应的命令。当把鼠标短时停留在按钮上时, 系统会给出工具提示 (Tooltip)。工具栏在主窗口中的位置见图 2.2, 它实际上就是提取了 Delphi 2005 最常用的菜单命令构成, 由此, 用户可以快速执行经常使用的操作和命令。

下面对这些常用的命令按钮进行说明, 工具栏如图 2.3 所示。

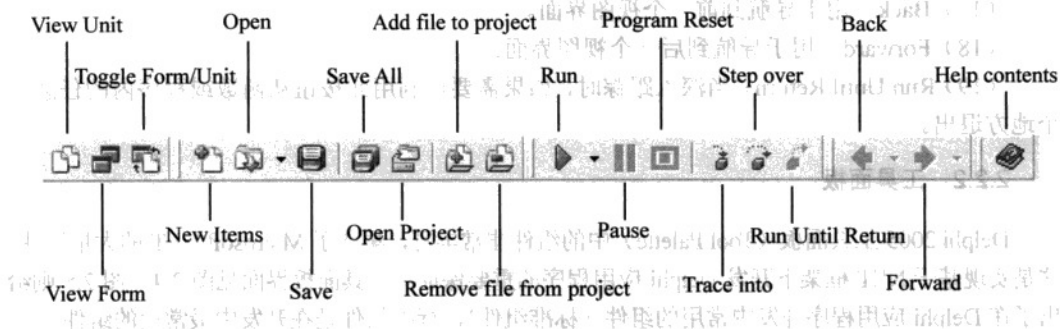


图 2.3 工具栏按钮分布

(1) View Unit。打开 View Unit 对话框, 其快捷方式为 Ctrl+F12。View Unit 对话框将显示当前项目的所有单元 (Unit) 列表, 用户根据需要可从中选择一个单元。

(2) View Form。用于打开 View Form 对话框, 其快捷方式为 Shift+F12。View Form 对话框将列出当前项目的所有窗体 (Form), 用户根据需要可从中选择一个窗体。

(3) Toggle Form/Unit。单击该按钮将实现与其相关的代码单元之间的切换, 其快捷方式为 F12。在实际的开发当中, 这种按钮会经常用到, 需要程序开发人员能熟练掌握。

(4) New Items。单击该按钮将弹出 New Items 对话框, 相当于选择菜单命令 File→New→other..., 表示要创建一个新的项目, 如创建一个 VCL 应用程序、一个 SDI 应用程序等, 这要取决于用户要在对话框中选择什么样的程序类型。

(5) Open。单击该按钮, 将打开 Open 对话框, 以供用户打开源文件或项目文件。

(6) Save。是一个保存按钮, 其快捷方式为 Ctrl+S。当文件代码被修改或刚创建时, 该按钮变为有效。快捷方式使用非常方便和快捷, 读者应非常熟悉, 它在编程过程会经常用得到。

(7) Save All。用于保存当前项目中所有打开的文件, 其快捷方式为 Shift+Ctrl+S。

(8) Open Project。用于打开一个已存在的项目文件, 其快捷方式为 Ctrl+F11。

(9) Add file to project。单击该按钮, 将弹出 Add to project 对话框, 用于向当前项目中添加一个已经存在的文件, 其快捷方式为 Shift+F11。

(10) Remove file from project。与 Add file to project 的功能相反, 其作用是从当前项目中

删除一个文件。

(11) Run。运行当前的应用程序，其快捷方式为 F9。这在编程调试过程中会经常用到，记住其快捷方式可有效地提高编程速度。

(12) Pause。用于暂停程序的执行，程序执行过程中有效。

(13) Program Reset。中断程序的执行，程序执行过程中有效，其快捷方式为 Ctrl+F2。

(14) Trace into。深入跟踪，即当前程序将逐句执行，会深入到函数或程序体内部执行，其快捷方式为 F7，在程序调试过程中有效。

(15) Step over。逐步跟踪，即当前程序将逐句执行，但它会把函数或程序体当作一行来处理，而不深入到内部执行，其快捷方式为 F8，在程序调试过程中有效。

(16) Help contents。用于启动电子文档帮助系统，有效地利用这些帮助，会极大地提高程序开发速度。

(17) Back。用于导航到前一个视图界面。

(18) Forward。用于导航到后一个视图界面。

(19) Run Until Return。当深入跟踪时，如果需要可利用此按钮从函数或程序内的任意一个地方退出。

2.2.2 工具面板

Delphi 2005 工具面板 (Tool Palette) 中的组件非常丰富，集成了 Microsoft.NET 的大量组件，这是实现基于 .NET 框架下开发 Delphi 应用程序的重要保证。工具面板界面见图 2.4，图 2.5 则给出了在 Delphi 应用程序开发中常用的组件 (标准组件)，标准组件是在开发中最常用的组件。

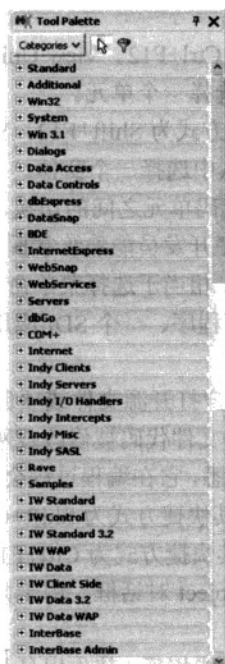


图 2.4 工具面板

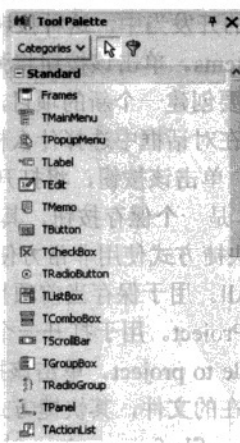


图 2.5 标准 (Standard) 组件

如果 IDE 没有工具面板, 可选择菜单命令 View→Tool Palette 来打开它, 其快捷方式为 Ctrl+Alt+P。

组件的使用方法较简单, 只要把需要的组件从工具面板中拖到窗体中即可, 但是组件的正确、合理使用对 Delphi 应用程序的开发至关重要, 本书将另立篇章予以重点讨论。

2.2.3 代码编辑器

代码编辑器也称为单元窗口, 它用于完成代码的编辑工作。Delphi 2005 代码编辑器是一个功能极为丰富的编辑器, 它可以提供多种错误检测功能和代码自动生成功能, 可以有效提高程序员的代码编写速度。

代码编辑器如图 2.6 所示, 如果出现多个代码单元, 该窗口将包含多个选项卡。单击某一个选项卡, 可以编辑相应单元文件的代码。

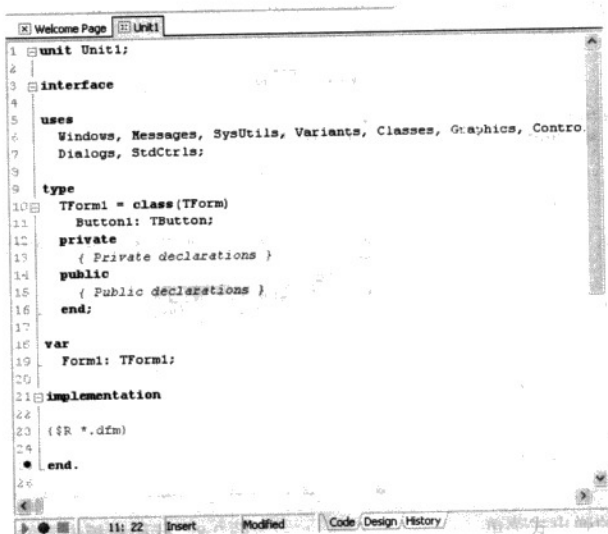


图 2.6 代码编辑器

以下分别介绍代码编辑器的主要功能。

1. 代码编辑

显然, 一个编辑器首先是一个文本工具, 所以它首先拥有类似文本编辑器一些相同的功能。例如, 可以使用光标控制键移动光标, insert 键可用于插入和替换模式的转换, 对不同功能的代码给予不同的显示颜色等。需要注意的是, 编辑器把任何超过 128 个字符的行都标记为错误, 这在字符串赋值时要当心。

2. 弹出式菜单

在编辑窗口(编辑代码的地方)上任意右键单击都会弹出如图 2.7 所示的快捷菜单。显然, 该菜单跟代码编辑有着紧密的联系, 充分地利用弹出式菜单的功能可以有效提高程序编写的速度、降低出错率等。了解菜单命令的功能只需在编辑窗口中多进行操作即可。

3. 代码洞察 (Code Insight)

Delphi 8、Delphi 2005 对这一部分进行了强化, 使其不断得到完善。它包含以下五个部分

的内容。

(1) 代码模板专家 (Code Templates Expert)。Delphi 将一些常用的代码结构做成模板保存起来, 构成模板专家。当程序员输入代码的时候, 有可能只记得某一代码结构的开头, 而忘记后面怎么写, 这时代码模板专家就派上用场了。例如, 当输入完 “for” 后, 忘记了后面是什么样的格式, 这时可以按下 Ctrl+J 组合键, 系统会给出如图 2.8 所示的帮助信息。当在弹出的框中双击相应的项, 如 “for statement”, 就会马上得到下列的 for 结构代码。显然, 除了上述功能外, 代码模板专家可以提高代码的编写速度等。

```
for := to do  
begin
```

```
end;
```

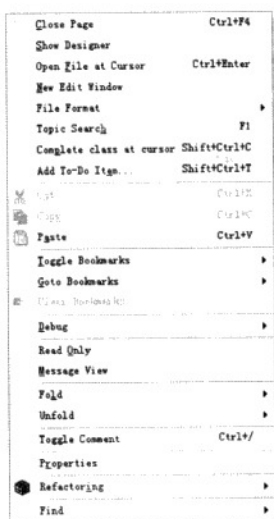


图 2.7 编辑窗口中的弹出式菜单

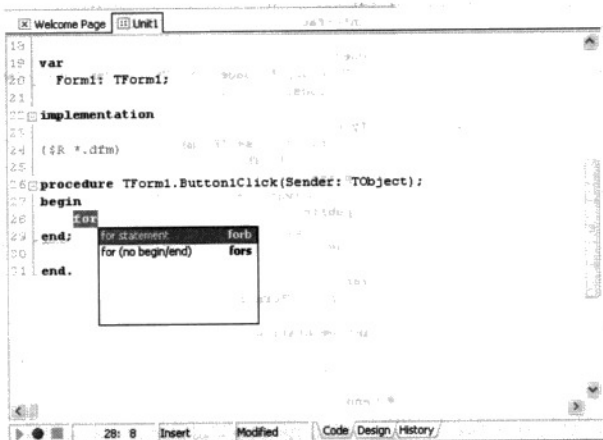


图 2.8 代码模板专家的应用

(2) 代码完成专家 (Code Completion Expert)。这个功能在代码编写过程中也是非常有用的, 其作用是在代码编写过程中可以自动找出对象的所有属性和方法, 供程序员作出快速的选择。另外, 在对象后面加上 “.” 号后, 系统会自动弹出一个框, 从中列出该对象所有的属性、方法等, 如图 2.9 所示。

(3) 代码参数专家 (Code Parameters Expert)。当输入函数 (或过程) 参数的时候, 如果参数很多, 程序员一般都很难记住它们的个数及其类型, 所以也就很容易弄错。这时代码参数专家可以扮演非常重要的角色: 自动提供参数个数和参数类型的帮助信息。例如, 在图 2.10 中当打完符号 “(” 后将给出函数参数的提示信息, 这样可以有效减少程序员写错函数或过程的机会。

如果不小心把提示框 “弄丢” 了, 可通过按组合键 “Ctrl+Shift+Space” 来重新获得。

(4) 表达式计算提示。用于查看任意一个变量的值, 主要用于调试。

(5) 符号提示。用于显示任意一个标识符的定义信息。其使用方法是, 在编辑器中把鼠标停留在待查看的标识符之上, 然后系统会显示该标识符的定义信息。例如, 把鼠标停留在标

识符 OnKeyDown 上, 将会产生如图 2.11 所示的提示信息。

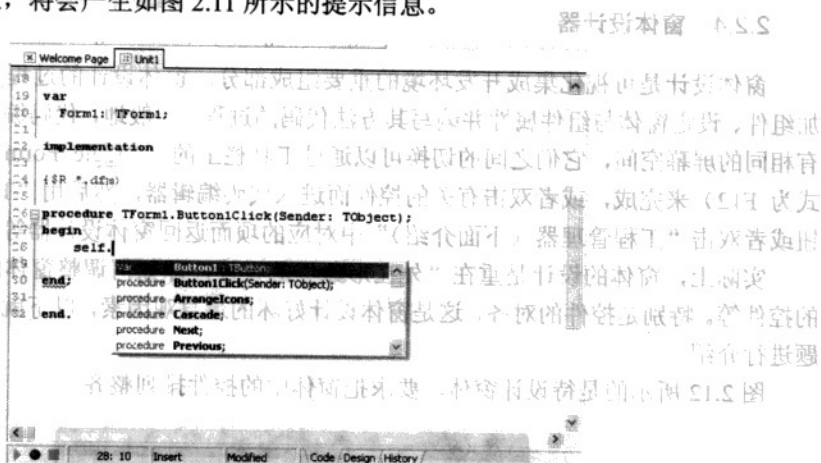


图 2.9 代码完成专家的应用

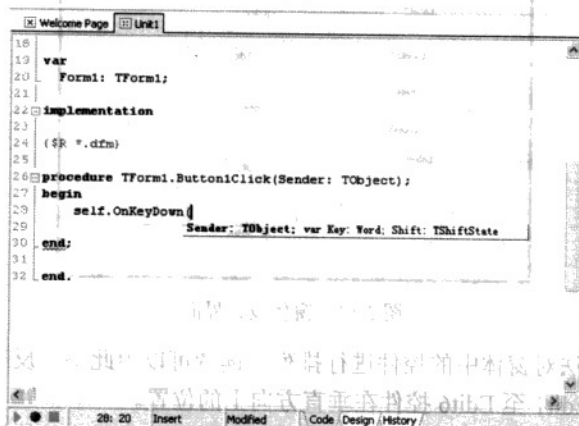


图 2.10 代码参数专家的应用

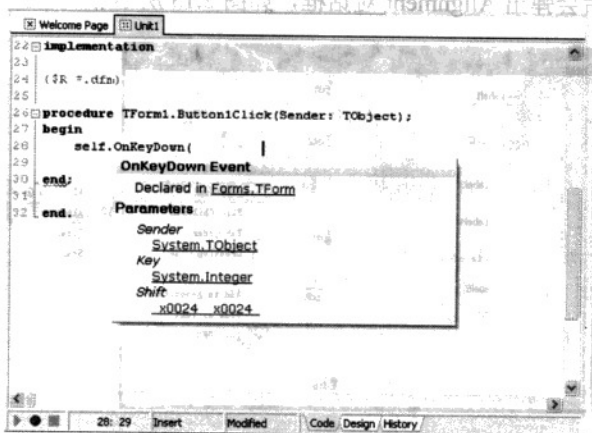


图 2.11 符号提示的应用

2.2.4 窗体设计器

窗体设计是可视化集成开发环境的重要组成部分。窗体设计的过程实际上是向窗体中添加组件、设定窗体与组件属性并编写其方法代码的过程。一般地，代码编辑器和窗体设计器占有相同的屏幕空间，它们之间的切换可以通过工具栏上的“Toggle Form/Unit”按钮（快捷方式为 F12）来完成，或者双击有关的控件而进入代码编辑器，然后用“Toggle Form/Unit”按钮或者双击“工程管理器（下面介绍）”中对应的项而返回窗体设计器等。

实际上，窗体的设计是重在“外在形象”的设计，即如何调整窗体的大小、对齐窗体上的控件等。特别是对齐，这是窗体设计好坏的最直观因素，下面就如何解决对齐控件问题进行介绍。

图 2.12 所示的是待设计窗体，要求把窗体中的控件排列整齐。

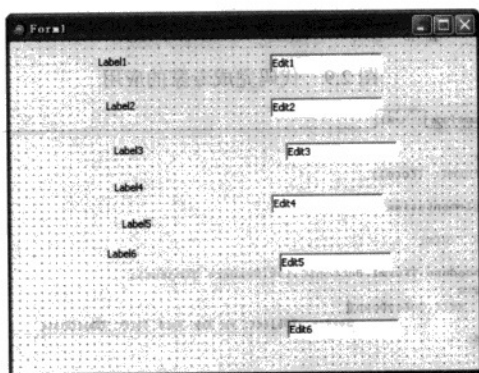


图 2.12 窗体设计界面

现在按照下列方法对窗体中的控件进行排列，读者可以由此举一反三。

(1) 首先确定 Edit1 至 Edit6 控件在垂直方向上的位置。

(2) 用鼠标选中所有的编辑框，并对 Edit 控件右键单击，在弹出的快捷菜单中选择 Position → Align...命令，然后会弹出 Alignment 对话框，如图 2.13 所示。

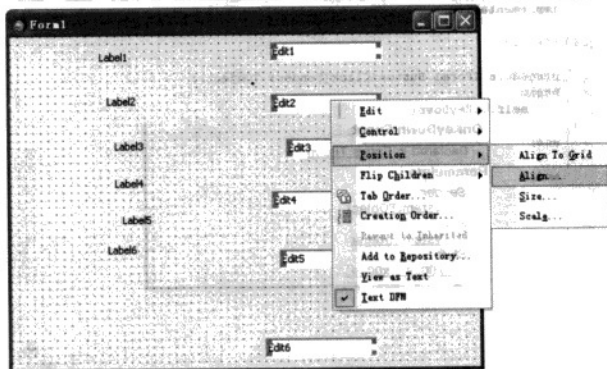


图 2.13 打开 Alignment 对话框

(3) 在 Alignment 对话框中, 分别设置水平和垂直方向上的对齐方式, 其中, 在水平方向上按靠右边对齐, 在垂直方向上等距离对齐(注意, Edit1 和 Edit6 在垂直方向上不会移动, 这也是为什么在第(1)中首先对它们定位的原因), 结果如图 2.14 所示。

(4) 单击 OK 按钮, 结果如图 2.15 所示。显然, 所有的 Edit 控件都对齐了。

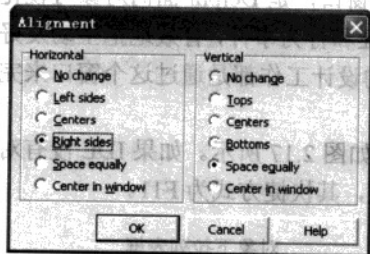


图 2.14 Alignment 对话框

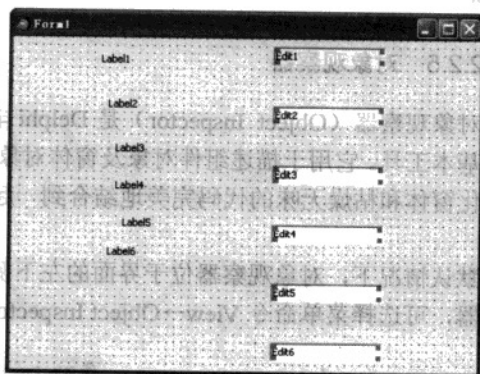


图 2.15 对 Edit 控件调整的结果

(5) 用鼠标调整 Label1 至 Label6 控件, 使之分别与 Edit1 至 Edit6 在水平方向上对齐, 然后用鼠标选定所有的 Label 控件, 并按照上述的方法将其对齐, 结果如图 2.16 所示。

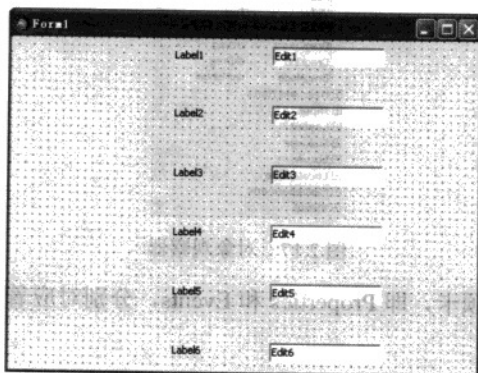


图 2.16 对控件的对齐结果

容易看出, 这种方法较手工调整来得快、准确和方便。当然, 不是在任何地方都可以采用上述方法, 有时还得用鼠标来移动控件。但在要求微调的地方, 往往鼠标也不适用, 那就得采用键盘来移动。Delphi 提供了“Ctrl 或 Shift+上、下、左、右键”的方法来分别微调控件的位置和大小, 精度为 1 个像素。其中, 一些组合快捷键说明如下。

- Ctrl+上键: 每按一次使被选中的控件向上移动 1 个像素。
- Ctrl+下键: 每按一次使被选中的控件向下移动 1 个像素。
- Ctrl+左键: 每按一次使被选中的控件向左移动 1 个像素。
- Ctrl+右键: 每按一次使被选中的控件向右移动 1 个像素。
- Shift+上键: 每按一次使被选中的控件的高度减 1 个像素。

- Shift+下键：每按一次使被选中的控件的高度加 1 个像素。
 - Shift+左键：每按一次使被选中的控件的宽度加 1 个像素。
 - Shift+右键：每按一次使被选中的控件的宽度减 1 个像素。
- 实际上，在应用中上述方法不是单独使用的，更多是混合运用，为设计好窗体而“各显神通”。

2.2.5 对象观察器

对象观察器 (Object Inspector) 是 Delphi 非常重要的窗口，是 Delphi 面向对象可视化编程的基本工具。它用于描述组件对象及窗体对象的属性特征和行为事件，有效地把生动友好的可视化窗体和枯燥无味的代码完美地结合到一起，大多数的设计工作都是通过这个窗口来完成的。

默认情况下，对象观察器位于界面的左下侧，其界面如图 2.17 所示。如果 IDE 没有对象观察器，可选择菜单命令 View→Object Inspector 来打开它，其快捷方式为 F11。

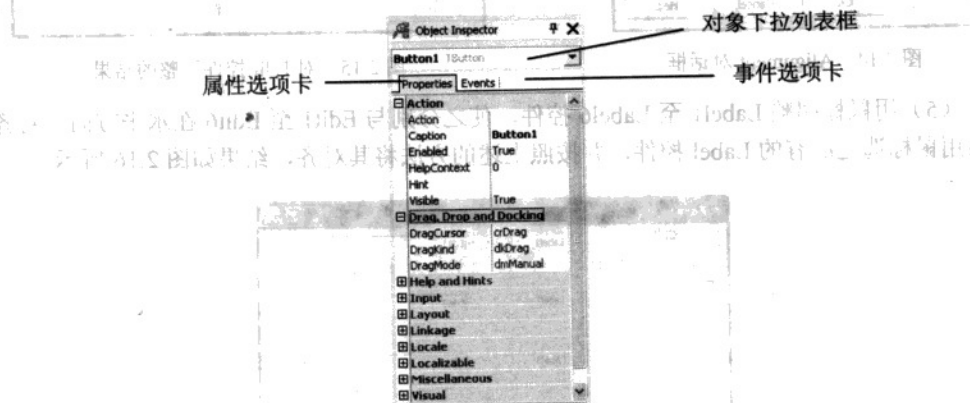


图 2.17 对象观察器

对象观察器有两个选项卡，即 Properties 和 Events，分别对应着对象属性和对象事件；还包含一个对象下拉列表框。

1. 对象下拉列表框

对象下拉列表框位于对象观察器的顶端，窗体及窗体中的一切组件都能在这里找到。单击列表框时会列出对象的名称和类型，供程序员选择使用。

2. 对象属性选项卡 (Properties)

在默认情况下，对象观察器显示对象属性选项卡对应的内容（否则可通过单击该选项卡的方法使之显示）。在此，程序员可以设计当前被选中对象的属性。有些属性左边有“+”号，这表示该属性还有子属性；有些属性右边有一个“...”，表示该属性要通过一个对话框来设置，即单击“...”按钮后会弹出相应的对话框，程序员要在该对话框中对属性进行设置。

如果窗体上有多个对象被选中，那么对象观察器将显示这些对象的共有属性，如果改变某一属性项的值，则所有被选中对象对应的属性值都将被改变。

不同的组件有不同的属性，但有些属性是许多组件共有的，在应用程序开发时经常用到，

熟练掌握这些属性的特性对程序的迅速开发起着非常重要的作用。表 2.1 介绍了一些组件常见的共有属性。

表 2.1 常用属性

属性名称	功能说明	属性名称	功能说明
Align	确定对象的对齐方式	Hint	设置提示信息
Caption	对象的标题	Left	左上角横坐标
ComponentIndex	对象的索引号	Name	对象的名字（编程时引用）
DragCursor	对象被拖动时鼠标的形状	ParentColor	确定是否引用父对象的颜色
DragMode	拖动操作方式	PopupMenu	确定对象上的弹出式菜单
Enabled	设定对象的可用性	TabOrder	确定用 Tab 键切换对象的次序
Font	对象的字体设定	Top	左上角纵坐标
Height	对象高度设定	Visible	确定对象的可见性
HelpContext	对象的帮助信息	With	对象的宽度

3. 对象事件选项卡（Events）

当单击事件选项卡时，对象观察器将显示被选中对象的所有事件。当双击其中的某一事件项左边的空白处时，Delphi 将自动建立一个事件处理函数编写的代码，并将焦点转移到代码编辑器，在此，程序员可以通过编辑代码来控制程序的行为，达到预定的目的。

如果直接双击，那么系统将为控件设置一个名称自动默认的函数来响应该事件；如果程序员先在事件项左边的空白处输入一个字符串，然后双击它，那么系统将以该字符串为函数名称来建立响应该事件的功能代码结构。

表 2.2 介绍了一些组件常见的共有事件。

表 2.2 常用事件

事件名称	功能说明	事件名称	功能说明
OnChange	对象发生改变时触发	OnKeyDown	按下键盘上某个键时触发
OnClick	单击鼠标左键时触发	OnKeyPress	按下键盘上某个 ASCII 码键时触发
OnDbClick	双击鼠标左键时触发	OnKeyUp	松开键盘上某个键时触发
OnDragDrop	其他对象拖入此对象时触发	OnMouseDown	单击鼠标时触发
OnDragOver	其他对象拖过此对象上方时触发	OnMouseMove	拖动鼠标时触发
OnEndDrag	拖动操作结束时触发	OnMouseUp	松开鼠标键时触发
OnEnter	焦点转移到对象时触发	OnResize	对象改变尺寸大小时触发
OnExit	焦点从对象转走时触发	OnStartDrop	拖动开始时触发
OnHide	对象隐藏时触发		

2.2.6 工程管理器

在默认情况下，工程管理器（Project Manager）位于 IDE 的右上角，其界面如图 2.18 所示。工程管理器以树状的形式把工程包含的所有单元都列出来，使得程序员对单元文件及其层

次结构一目了然。另外,工程管理器还可以通过工程组对工程进行管理,列出了反映各工程之间联系的树状表,使得程序员可以非常直观地了解各工程之间的关系,方便将相互联系的工程中共同的动态链接库文件与可执行文件组织到一起。

工程管理器的上部还有三个图形按钮,它们的作用如下:

- **Activate** 按钮用于激活一个工程(当有多个工程时)。
- **New** 按钮用于在工程管理器中添加一个工程,其作用相当于选择菜单命令 **File→New→Other...**。
- **Remove** 按钮用于把工程管理器中当前被激活的工程删除。

工程管理器的打开方法是,选择菜单命令 **View→Project Manager**,或者用快捷方式 **Ctrl+Alt+F11**。

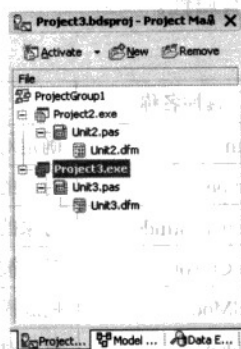


图 2.18 工程管理器

2.2.7 对象树浏览器

对象树浏览器(Structure)位于 IDE 的左上方,其界面如图 2.19 所示。它以树状的形式显示窗体、数据模块或框架上所有组件(可视的和非可视的)之间的逻辑关系。

如果 IDE 没有对象树浏览器,可选择菜单命令 **View→Structure** 来打开它,其快捷方式为 **Alt+Shift+F11**。

2.2.8 模型视图

在 IDE 中,模型视图(Model View)与工程管理器占用同一个屏幕空间,但单击 Model View 选项卡可以显示该视图,如图 2.20 所示。

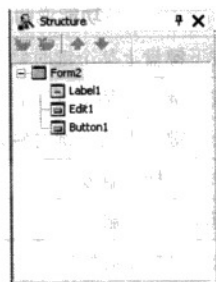


图 2.19 对象树浏览器

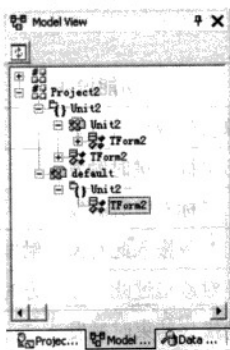


图 2.20 模型视图

模型视图主要有下面两个作用:

(1) 以树状形式显示出工程中每个类的结构图,包括类的属性、方法和事件等。通过双击模型视图中的节点可以快速地定位到代码编辑器中相应的代码位置。

(2) 在模型视图中的单元节点上单击右键,在弹出的快捷菜单中选择 **Select on Diagram** 命令,则将进入 UML 类的建模环境,可以方便实现对类等的可视化操作,如图 2.21 所示。

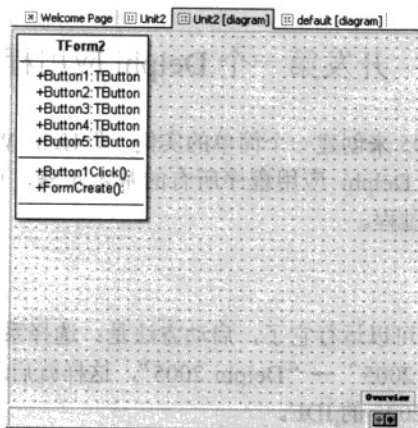



图 2.21 UML 类建模环境

2.2.9 帮助系统

Delphi 2005 提供了完全的 .NET 平台下的开发帮助，具有强大的帮助功能。

可以选择菜单命令 **Help**→**Borland Help**，或者单击工具栏上的快捷按钮  (**Help contents**)，来打开 Delphi 2005 的帮助系统，图 2.22 所示是 Delphi 2005 帮助系统的界面。

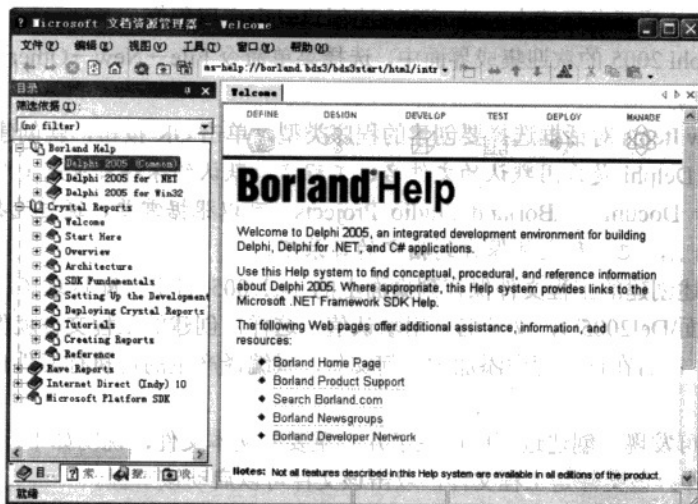


图 2.22 Delphi 2005 帮助系统的界面

通过帮助系统，程序员可以非常方便地通过搜索、索引、筛选等途径找到相关的内容和资料，也可运用菜单或工具栏上的快捷命令进行协助，还可以运用树形目录快速找到相应的帮助内容等。

此外，在代码编辑器，系统也可以提供实时的在线帮助，这些帮助都非常有效。具体操作见上一节的介绍。

2.3 开发第一个 Delphi 应用程序

本节主要利用 Delphi 2005 来创建一个简单的实例,即 Hello World 程序实例。其功能虽然非常简单,但是包含了开发 Delphi 应用程序所有的基本步骤,可以让读者了解或者回顾用 Delphi 2005 开发程序的基本过程。

2.3.1 创建工程

安装完 Delphi 2005 后就可以运行它了,启动方法是:选择系统菜单命令“开始”→“所有程序”→“Borland Delphi 2005”→“Delphi 2005”,这样就启动了 Delphi 2005,出现启动装载界面,最后打开 Delphi 2005 的 IDE。

Delphi 2005 的启动过程比较长,大概需要 1 分钟左右。为了快速启动 Delphi 2005,请按下列方法对系统进行设置:右键单击系统菜单中的命令“开始”→“所有程序”→“Borland Delphi 2005”→“Delphi 2005”,在弹出的菜单中选择“属性”项,然后在打开的“Delphi 2005 属性”对话框中选择“快捷方式”选项卡,最后在“目标”文本框中的字符串后面加空格,再加上 `/ns` 作为命令参数,单击“确定”按钮即可。笔者曾就此做过实验,发现设置前和设置后,Delphi 2005 启动所需的时间分别为 55 秒和 21 秒。显然,设置后所需的时间较短,但启动过程中不出现任何提示画面,这需要注意!

在 Delphi 2005 的开发环境中,按下列方法创建一个应用程序:

(1) 在 Delphi 2005 的欢迎集成界面中,选择菜单命令 `File→New→Other...`,然后会弹出 New Items 对话框。

(2) 在 New Items 对话框选择要创建的程序类型,单击 OK 按钮,就创建了新的工程。

注意:这时 Delphi 是采用默认的文件名、工程名,默认的保存目录是 `C:\Documents and Settings\meng\My Documents\Borland Studio Projects`。可以根据需要,通过选择菜单命令 `File→Save Project As...`,把工程文件保存到指定的目录下。

这里,把上述创建的工程文件保存到桌面上的 Del2005 目下(即目录 `C:\Documents and Settings\meng\桌面\Del2005\`),文件名采用默认值。这样,创建的工程文件就保存到桌面上的 Del2005 目下了。以后在该工程中添加的任何文件,或编译产生的可执行文件也会自动保存到该目录下。

查看该目录可发现,创建过程中产生了几个重要的文本文件,说明如下:

- .dpr 文件:该文件是工程文件,双击该文件可以启动该工程。
- .dfm 文件:用于保存窗体上对象及其属性的设置。
- .pas 文件:用于保存窗体对应的单元源代码文件。

2.3.2 窗体设计

下列作一个简单的设计,以便于加深读者对窗体设计的认识,读者可以在学习中慢慢体会,举一反三。

(1) 从工具面板中把 TLabel 组件拖到窗体中,用鼠标适当调整其大小。

(2) 在对象查看器中,选择属性选项卡,展开 Miscellaneous 属性项,找到 Name 这个子

属性项，并将其改为 LabHello，即把刚添加的 TLabel 组件对象命名为 LabHello。

(3) 用与 (2) 类似的操作方法，把属性对象 LabHello 的 Caption 属性值设为空，Alignment 属性值设为 taCenter（使得文字显示时居中），把 Font 属性项下的子属性 Size 设为 30（设定字体大小）。

(4) 从工具面板中把 TButton 组件拖至窗体中。

(5) 在对象查看器中，选择属性选项卡，将 Caption 属性值改为 ShowHello，以改变按钮的字体显示。

(6) 适当调整窗体上对象的位置和大小，使之整齐美观。

(7) 单击选择窗体，按照 (4) 的方法把窗体的 Caption 属性值改为“我的第一程序——Hello World!”

这样，就把这个简单程序的界面设计好了，结果如图 2.23 所示。下面为其添加相应的代码，使之能“动”起来。



图 2.23 窗体设计结果

在把组件从工具面板中拖到窗体上时，有时需要多次拖动同一组件，这显得很繁琐。一种解决方法就是：按 Ctrl 键的同时单击工具面板中需要的组件，然后在窗体中多次单击不同的地方（这时可以放开 Ctrl 键），每单击一次都会产生该组件类型的对象，这样就不需要多次的拖动操作。

2.3.3 代码编写

在窗体中选中的 TButton 按钮，打开对象观察器的事件选项卡，把 OnClick 项的值改为 btShowHello，即把响应事件的函数名改为 btShowHello，并双击该值，则进入代码编辑窗口（代码编辑器），代码编辑器会自动创建并显示该函数，然后添加如下代码：

```
procedure TForm4.btShowHello(Sender: TObject);  
begin
```

```
    LabHello.Caption := 'Hello World!';
```


```
end;
```

当然，可以直接双击窗体上的 TButton 按钮进入代码编辑器，但是这时代码编辑器会采用默认的函数名——Button1Click。如果程序代码很多的时候，其可读性是很差的，所以建议采用具有相应意义、方便记忆的函数。

至此，代码编辑器中的全部代码如下：

```
unit Unit4;  
interface
```


```
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;
type
    TForm4 = class(TForm)
        LabHello: TLabel;
        Button1: TButton;
        Button2: TButton;
        procedure Button2Click(Sender: TObject);
        procedure btShowHello(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
var
    Form4: TForm4;
implementation
    {$R *.dfm}
    procedure TForm4.btShowHello(Sender: TObject);
    begin
        LabHello.Caption := 'Hello World!';
    end;
```

代码编写完了以后一定要记得保存，以免程序的丢失。保存工程是指保存所有的工程文件，包括.dpr、.pas 和.dfm 文件，方法是：单击工具栏上的快捷按钮  (Save All)，这时弹出的第一个对话框是保存单元文件 Units6.pas 的对话框，这里采用默认名 Units6.pas，单击“保存”按钮。

之后，会弹出第二个用于保存工程文件的对话框，这里把工程名改为比较直观的名字：HelloWorldProject，然后单击“保存”按钮。至此，工程文件保存完毕。

每次创建一个工程文件，都要把它保存下来，以备后用。

2.3.4 实例运行

确保代码的正确性后，就可以运行程序了。其方法是：单击工具栏上的  (Run) 按钮，则开始编译程序。如果代码中有错误，则代码编辑器会给出错误的原因和出错的地方，以让程序员修改；如果代码正确，那么编译完后将直接运行产生的.exe 文件。

上述程序的运行结果如图 2.24 所示。



图 2.24 Hello World 程序的运行结果

通过这个例子可以看出，在 Delphi 2005 中编写一个应用程序基本包括以下 4 个步骤。

- (1) 建立一个新的工程，或打开一个已经存在的工程。
- (2) 向窗体上添加需要的控件并修改控件的属性，使得窗体整体外观符合用户的要求。
- (3) 编写响应事件处理的函数或过程，包括添加用户定义的过程和函数。
- (4) 调试、编译并最终运行程序。

2.4 小结

本章首先介绍了 Delphi 2005 的可视化集成环境的基本组成部分及其主要功能；然后介绍了 Delphi 2005 的可视化集成环境的使用方法，在介绍过程中融入了作者多年积累的一些经验和结论，以加深读者对 Delphi 2005 开发环境的认识和掌握；最后具体介绍了一个 Delphi 程序实例的开发步骤，让读者对基于 Delphi 2005 的开发方法有一个感性认识。通过对本章的学习，应该掌握以下内容：

- 创建一个高效的 Delphi 2005 的可视化集成环境 (IDE)。
- 对 Delphi 2005 的可视化集成开发环境有一个深刻的认识，并能熟练使用，包括以下三个方面：
 - 熟悉主窗口和工具面板；
 - 熟练使用编辑器编写代码；
 - 掌握窗体设计器、对象观察器、工程管理器、对象树浏览器及其模型视图的基本运用。
- 掌握开发一个应用程序的基本步骤。

第3章 熟悉 Object Pascal 语言的编程技术

Object Pascal 是 Delphi 的编程语言,它是在 Borland 公司赖以发家的 Pascal 语言的基础上引入 OOP 技术而形成一门编程语言。学好 Object Pascal 是基于 Delphi 应用开发的基础。本章中,将对 Object Pascal 程序设计的基本语法特征进行介绍,同时赋予大量而具体的例子,并对 Delphi 中基于 OOP 技术的编程方法进行了介绍。本章涉及的内容要点包括:

- Object Pascal 的特点。
- Object Pascal 语言的基本语法,包括其数据类型、基本运算、流程控制语句的运用。
- Object Pascal 过程和函数的定义和调用。
- Object Pascal 面向对象的编程技术,包括对象与类的概念、类的声明和对象的创建和撤销方法,以及类的封装性、继承性和多态性。

3.1 Object Pascal 语言简介

3.1.1 什么是 Object Pascal

Delphi 以 Object Pascal 为编程语言,该语言是由 Borland 公司赖以发家的 Pascal 语言发展而来的。Pascal 语言是于 20 世纪 70 年代开发出的一种语法严谨、条理清晰的结构化程序设计语言。使用 Pascal 语言编写的程序具有可读性强、结构紧凑等特点。因此,许多高校和科研院所都一直把它作为计算机专业的教学语言,以培养开发者良好的结构化程序设计习惯。

随着面向对象技术的迅速发展和成熟,面对 Visual Basic 的凶猛来势和强大的竞争力,Borland 公司沉着应战,在 Pascal 语言基础上引入了面向对象技术,开发出了集 Pascal 语言优点和面向对象技术于一身的 Object Pascal 语言。Object Pascal 语言在一定程度上超越了 Visual Basic,具有面向对象的功能,编译速度快(甚至优于 VC++),能够进行可视化开发,同时拥有 Windows Framework 组件架构。基于 Object Pascal 编程语言的 Delphi 已经成了 Visual Basic 杀手的美名。

Object Pascal 语言是一种复杂的语言,详细的论述足以写成一本书,而仅仅用一章是无法讲清楚的,所以在本章中主要介绍其中编程常用的、内容最基本的部分。

3.1.2 Delphi 文件及其结构分析

在 Delphi 中,一个工程(Project)包含三类主要文件,即工程文件、窗体文件和单元文件。以下分别加以说明。

1. 单元文件

单元文件都是以.pas 为扩展名的源程序代码文件,最常见的单元文件是与窗体为对象的单元文件,另外还有用于存储函数和过程的单元文件及用于制作组件的单元文件,这两类单元文件都是与窗体没有直接联系的。

打开 2.3 节创建的工程 HelloWorldProject, 在编辑器中打开单元文件 Unit4, 其代码如图 3.1 所示。

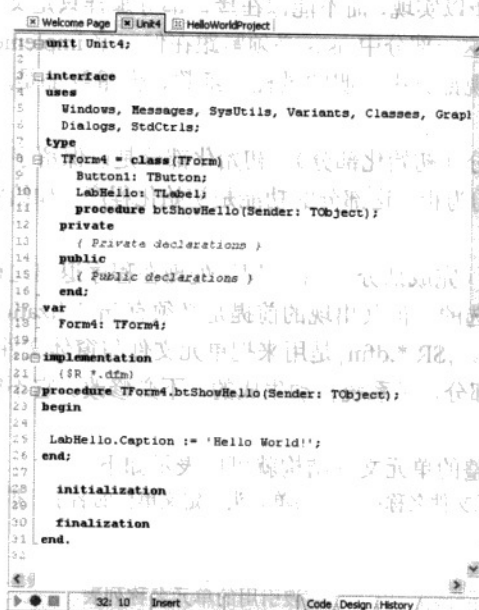


图 3.1 单元文件 Unit4.pas 的代码

从这段代码中可以看出, 单元文件由以下几部分组成:

(1) 单元头。用于定义单元的名称, 格式为:

unit 单元名称;

(2) interface 部分 (接口部分)。用于定义单元的接口, 格式为:

interface

uses

// 被引用的程序单元列表

var

// 声明变量、数据类型、常量、过程和函数等

接口部分是从保留字 interface 开始一直到保留字 implementation 之前的一条语句。在接口部分中可以包含 uses 语句、var 语句和 type 语句等。uses 语句用来引入需要的程序单元, 如在单元 Unit4 中被引入的程序单元包括 Windows、Messages、SysUtils、Variants、Classes、Graphics、Controls、Forms、Dialogs、StdCtrls 等, 但它必须紧跟在 interface 之后, 即在 interface 和 uses 之间不能有其他语句; var 语句则用于声明变量、数据类型、常量、过程和函数等, 在单元 Unit4 中则定义了类型为 TForm4 的变量 Form4; 而 type 语句可以让程序员定义自己的数据类型, 上例中定义了 TForm4 类, class(TForm) 表示它是 TForm 类的派生类。该类包含了一个 TButton 类对象和 TLabel 类对象, 以及一个用于处理 TButton 类对象事件的函数 btShowHello。

在接口部分声明的变量、数据类型、常量、过程和函数等, 在整个程序中都可以访问, 但是单元的 uses 语句的列表中要包含被引用单元的名称。

(3) implementation 部分 (单元的实现部分)。单元的实现部分是从保留字 implementation

开始一直到 initialization 之前的一条语句。在接口部分定义的过程、函数等必须在这一部分予以实现。另外,实现部分也可以拥有自己的 uses 语句和 var 语句等,但是在这部分中定义的函数和过程也必须在这部分予以实现,而不能像在接口部分那样只定义过程或函数名。与在接口部分的 uses 语句类似,在这一部分中 uses 必须紧跟在保留字 implementation 之后。

需要注意的是,在实现部分中声明的过程、函数、变量等都是局部的,不能被其他单元引用。

(4) initialization 部分(初始化部分)。初始化部分是以保留字 initialization 开头,直到 finalization 之前的一条语句为止。这部分的功能是初始化程序,如给定义的变量赋初值、开辟内存单元等。


(5) finalization 部分(完成部分)。主要用于处理在程序退出之前的最后工作,以释放系统资源等。这部分也是可选的,但其出现的前提是必须有 initialization 部分。

(6) {\$R *.dfm} 部分。{\$R *.dfm} 是用来把单元文件与窗体文件(.dfm 文件)连接起来,一般放在 implementation 部分,是系统自动生成的,不必修改。它不能被删除,否则会引起编译错误。

这样,一个标准而完整的单元文件结构就可以表示如下:

	Unit 单元文件名称;	//单元头,定义单位的名字,扩展名为.pas
接口部分	Interface	
	uses	
	type	被引用的单元名称列表
	var	被定义的数据类型 //变量定义
实现部分	implementation	//接口实现部分
	{\$R *.dfm}	//建立窗体和单元的连接
	...	
初始化部分	initialization	//初始化部分,可选
	...	
完成部分	finalization	//完成部分,可选
	...	
	end.	//单元文件结束(以“.”为标志)

2. 窗体文件

窗体文件是.dfm 文件,它用来描述窗体的属性,如高(Height)、宽(Width)等。每一个窗体都有一个单元文件与之对应,而且它们是同名的,只是扩展名不同而已。例如,对于上例中,由于把窗体对应的单元文件命名为 Unit4.pas,所以该窗体文件就自动设置为 Unit4.dfm。窗体文件可以用记事本打开,也可以在代码编辑器中打开,方法是:单击工具栏上的  按钮(Open),将弹出 Open 对话框中,然后在“文件类型”下拉列表框中选择 Any file (*.*) ,在“文件名”下拉列表框中选择文件 Unit4.dfm,最后单击“打开”按钮,即可在代码编辑器中打开窗体文件 Unit4.dfm,如图 3.2 所示。

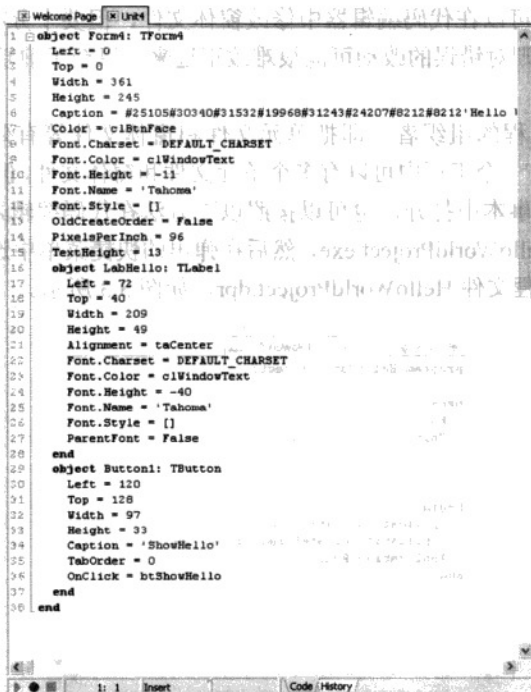


图 3.2 窗体文件 Unit4.dfm 的代码

从图 3.2 中可以看出窗体的各种属性,包括各对象属性的描述等。描述是按级别方式来展开的,首先是窗体本身属性的描述(从“object Form4: TForm4”开始),然后分别以 object ... end 为代码段对 TLabel 对象和 TButton 对象进行描述。

于是,一个窗体文件的结构就可以表示如下:

```

object 窗体名称: 窗体类
...
窗体属性描述 {
...
...
object 对象 1: 对象 1 的类型
...
对象 1 属性描述 {
...
...
end
object 对象 2: 对象 2 的类型
...
对象 2 属性描述 {
...
...
end
其他对象的属性描述 {
...
end

```

当然，程序员完全可以在代码编辑器中修改窗体文件，但要十分小心。对于初学者，建议不要在这里修改，否则对错误的改动可能很难改正过来，以至于前功尽弃。

3. 工程文件

工程文件是整个工程的组织者，即把单元文件和窗体文件等有效地连接在一起，它是以.dpr 为扩展名的文件。一个工程中可以有多个单元文件和窗体文件，但只能有一个工程文件。

工程文件可以在记事本中打开，也可以按照以下方法在代码编辑器中打开：在工程管理器中右键单击工程名 HelloWorldProject.exe，然后在弹出的快捷菜单中选择 View Source，即可在代码编辑器中打开工程文件 HelloWorldProject.dpr，如图 3.3 所示。

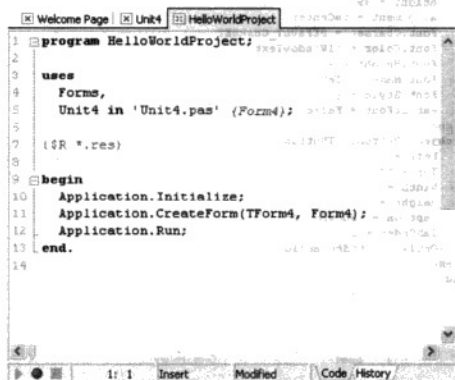


图 3.3 打开工程文件 HelloWorldProject.dpr

由图 3.3 可见，工程文件分为以下三部分：

(1) 程序头。程序头是用于指定工程的名称。图 3.3 表明该工程名为 HelloWorldProject，编译后将生成名为 HelloWorldProject.exe 可执行文件。

(2) uses 部分。该部分说明在工程中要引用的单元，只要把待引用的单元名称以逗号隔开列在保留字 uses 的后面即可。

(3) begin ... end 部分（执行部分）。该部分是程序执行的入口处，包含了执行需要的语句，其中 Application.Initialize 用于程序初始化，函数 CreateForm() 则用于创建窗体对象，Application.Run 使程序真正运行起来。

(4) {\$R *.res} 命令。该命令用于把资源文件（二进制文件）粘贴到已经编译过的可执行文件中，是不可或缺的。

由以上分析，工程文件的结构可以抽象如下：

```
program 工程名称;
uses
  ...
  {$R *.res}
begin
  Application.Initialize;
  ...
  Application.Run;
end.
```

3.2 熟悉 Object Pascal 的语法要素

3.2.1 Object Pascal 字符集与标识符

1. 字符集

Object Pascal 程序使用的是 ASCII 码字符集, 包含如下字符:

大小写字母: A...Z 和 a...z;

数字: 0...9;

符号: ()+-*/<>=!~^;:.'@%,"#\$%&_{}?[];

Tab 键, 空格键, 回车键。

Object Pascal 对大小写不敏感, 如 MyName 与 myname 对 Object Pascal 来说是相同的。

2. 标识符

在编写 Object Pascal 程序时需要构造大量的“词”——标识符, 它是程序这座大厦的“砖块”, 在常量、变量、函数和过程等命名时都要用到它。标识符是由字母、数字、下画线(_)组合而构成的字符串, 其最大长度不能超过 255 个字节。另外, 标识符只能以字母或下划线打头, 而不能以数字开头, 不能包含连字号、斜线号、空格等字符。从以下举的几个例子中, 可以体会到构造标识符的一些规律。

下列的标识符是正确的:

_Y22

Y22_

Name

stu_namet

下列的标识符则是错误的:

delphi&ID

delphi-ID

delphi /ID

delphi 2005

delphi\$ID

2005Delphi

这是因为它们分别使用了非法字符&、-、/和空格符, 以及以数字打头等。另外, Object Pascal 对大小写不敏感, 所以在标识符中大小写可以混用, 如 DelphiID 与 delphiid 是等价的。当然, 在字符串中大小写是有区别的。

标识符的书写不应过于简单(如 mm), 而应该根据实际开发的需要及开发人员的记忆习惯来写, 如 Student_ID 等, 以便于开发人员理解。

3.2.2 Object Pascal 保留字

保留字是 Object Pascal 预留的标识符, 只能由 Object Pascal “使用”, 而用户就不能定义与保留字相同的标识符。例如, 下面的语句是错误的:

```
var and:integer;
```


因为 and 是保留字，所以不能把它用作变量名。Object Pascal 包含的保留字如表 3.1 所示。

表 3.1 Object Pascal 包含的保留字

and	exports	mod	shr
array	file	nil	string
as	finalization	not	then
asm	finally	object	threadvar
begin	for	of	to
case	function	or	try
class	goto	out	type
const	if	packed	unit
constructor	implementation	procedure	until
destructor	initialization	program	uses
dispinterface	inherited	property	var
div	in	raise	while
do	inline	record	with
downto	interface	repeat	xor
else	is	resourcestring	
end	label	set	
except	library	shl	

3.2.3 常量与变量

1. 常量

常量是指在程序运行过程中，其值不能改变的量。常量有不同的类型（关于数据类型将在下文介绍），例如，3.14 为数字常量，“I am a Chinese.” 为字符串常量等。在编程过程中，更多时候是用一个标识符来代表一个常量。在 Object Pascal 中常量是用关键字 const 来声明标识符的。例如，分别用 PI 和 PERSON 代表常量 3.14 和常量 “I am a Chinese.”，则可以用下列的语法：

```
const
  PI=3.14;
  PERSON='I am a Chinese.';
```

从上面的语句中可以看出，虽然常量具有数据类型，但在声明时并不显示说明常量的数据类型。实际上，编译器是根据常量的值来识别它的类型，并分配相应的存储空间。

2. 变量

在程序运行过程中，其值能够改变的量称为变量。在形式上，变量是用关键字 var 来声明的标识符。声明的格式为：

```
var 变量: 数据类型;
```

例如，分别声明整型变量和字符串变量 total、name，并赋值 100 和 '张三丰'，则可用下列语句：

```

var
    total:integer;
    name:string;
begin
    total := 100;
    name := '张三丰';
end;

```

3. 变量与常量的区别

在作用域中，常量不能重新被赋值，而变量则可以重新赋值这也是变量的意义所在，因为变量是程序代码中代表一个内存地址的标识符，而相应的内存单元的值是可以改变的。例如，假设 PI 是上述定义的常量，name 为上述定义的变量，则在下列语句中：

```

PI := 3.14159;
name := '张无忌';

```

第一个语句是错误的，第二个是正确的，因为 PI 为常量，其值不能改变；name 为变量，可重新赋值。

3.2.4 运算符

按类型分，Object Pascal 中的运算符有以下几种：赋值运算符、算术运算符、逻辑运算符、关系运算符、字符串运算符、位运算符、地址和指针运算符等。具体的运算符见表 3.2。

表 3.2 Object Pascal 的运算符

运算符类型	操作符	说明
赋值运算符	:=	把一个常量或变量的值赋给另一个变量
算术运算符	+	两数相加，也有单目运算的形式
	-	两数相减，也有单目运算的形式
	*	两数相乘
	/	两浮点数相除
	div	两整型数相除，结果返回商（整型）
	mod	两整数取模
逻辑运算符	not	逻辑非
	and	逻辑与
	or	逻辑或
	xor	逻辑异或
关系运算符	=	判断是否相等，相等则返回 true，否则返回 false
	<>	判断是否不相等，不相等则返回 true，否则返回 false
	<=	判断是否小于或等于，是则返回 true，否则返回 false
	>=	判断是否大于或等于，是则返回 true，否则返回 false
	>	判断是否大于，是则返回 true，否则返回 false
	<	判断是否小于，是则返回 true，否则返回 false
	in	对于 A in B，如果 A ⊆ B，则返回 true，否则返回 false

续表

运算符类型	操作符	说明
字符串运算符	+	将两个字符串连在一起形成一个字符串
位运算符	not	按位取反
	and	按位取与
	or	按位取或
	xor	按位取异或
	shl	按位左移
	shr	按位右移
地址和指针运算符	+	使指针增加一个偏移量
	-	使指针减少一个偏移量
	^	取指针所指向的单元的内容(值)
	=	判断两个指针是否相等(指向同一个单元地址), 相等则返回 true, 否则返回 false
	<>	判断两个指针是否不相等, 不相等则返回 true, 否则返回 false
	@	取变量的地址

3.2.5 注释符

在高级程序设计语言中都有注释符, 但不同的语言其注释符也不尽相同。注释符的作用是对程序代码的解释和说明, 不参加编译。养成良好的注释习惯对一个程序员来说是非常必要的, 特别是在大型的程序开发中, 对开发本人或其他维护人员而言都是非常重要的。

Object Pascal 语言支持三种注释方式:

- (1) {被注释的部分};
- (2) (*被注释的部分*);
- (3) //被注释的部分。

其中, (1) 和 (2) 中的注释方式可以对多行注释, 但不支持嵌套注释, 而 (3) 中的方式只能注释一行。

“{”或“(”之后紧跟美元符号“\$”时, 表示的不是注释, 而是编译指令。例如, {\$APPTYPE CONSOLE}就是一条编译指令, 它说明该程序是一个控制台应用程序。

3.3 熟悉 Object Pascal 的数据类型

数据类型是高级程序设计语言中重要的语法要素, 掌握各种不同数据类型的特点是学好程序设计语言的基础。Object Pascal 是在 Pascal 语言的基础之上发展起来的, 它继承并丰富了 Pascal 语言的数据类型, 可以在 Object Pascal 中构造各种复杂的数据类型。Object Pascal 提供的数据类型主要包括六种, 以下分别给予说明。

3.3.1 简单数据类型

简单数据类型是 Object Pascal 中最基本的数据类型，它又可以分为两种，即有序类型和实数类型。

1. 有序类型

有序类型包括整数类型、字符类型、布尔类型、枚举类型和子界类型。实际上，该数据类型把数据定义成一个有次序的数值的集合，该集合中任意一个数值都有惟一的前驱者（除非是集合中的第一个数值）和惟一的后继者（除非是集合中的最后一个数值）。对集合中的每一个值都有对应的一个序数标识它在集合中的位置，默认情况下，集合中第一个值的序数为 0、第二个为 1，依此类推。Object Pascal 中提供了获取这种序数的函数，包括 Ord、Pred、Succ、High、Low 等五个函数，其具体说明见表 3.3。

表 3.3 有序数据类型的常用函数

函数	参数	返回	举例说明
Ord	序数类型表达式	序数值	Ord('A')=65, Ord(8)=8（但不能用于 Int64 类型）
Pred	序数类型表达式	前驱	Pred('b')='a', Pred(2)=1
Succ	序数类型表达式	后继	Succ('a')='b', Succ(2)=3
High	序数类型变量	该类型的最大序数值	High(Byte)=255, High(integer)=2147483647
Low	序数类型变量	该类型的最小序数值	Low(Byte)=0, Low(integer)=-2147483648

以下分别对有序数据类型的各种类型进行介绍。

(1) 整数类型。整数类型是编程中最常用的一种数据类型，在 Object Pascal 中，它可以分为通用整数类型和基本整数类型。

1) 通用整数类型。通用整数类型包括 Integer 和 Cardinal 两种。在编程中如需要用整数的地方就尽量使用这两种，因为它们具有一些优化能力，使得运算速度快、消耗资源少。

Integer 和 Cardinal 的区别在于，前者表示有符号的整数，后者则表示无符号的整数。它们的数值范围和长度会随着 CPU 和操作系统的不同而不同。用 High 函数可以测出 Integer 和 Cardinal 在 32 位 Object Pascal 编译器中的数值范围，如表 3.4 所示。

表 3.4 通用整数类型的数值范围和大小

通用类型	数值范围	大小
Integer	-2 147 483 648 ~ 2 147 483 647	32 位（有符号）
Cardinal	0 ~ 4 294 967 295	32 位（无符号）

2) 基本整数类型。基本整数类型包括 Shortint、Smallint、Longint、Byte、Word、LongWord、Int64，它们的范围、大小如表 3.5 所示。

范围越大、长度越长、存储能力越强的类型所需的内存资源也就越大，所以在实际应用中需要对定义的对象进行估计，采用适当的类型。例如，如果一个变量的数值活动范围在 0~200 之间，则应采用 Byte 类型，而如果用 LongWord 则变成“杀鸡用牛刀”了——极度浪费系

统资源。

表 3.5 基本整数类型的数值范围和大小

通用类型	数值范围	大小
Shortint	-128 ~ 127	8 位 (有符号)
Smallint	-32 768 ~ 32 767	16 位 (有符号)
Longint	-2 147 483 648 ~ 2 147 483 647	32 位 (有符号)
Byte	0 ~ 255	8 位 (无符号)
Word	0 ~ 65 535	16 位 (无符号)
LongWord	0 ~ 4 294 967 295	32 位 (无符号)
Int64	$-2^{63} \sim 2^{63}-1$	64 位 (有符号)

(2) 字符类型。字符类型包含 AnsiChar 和 WideChar 类型。AnsiChar 是 8 位字符集, 存储一个 Ansi 字符, 因而 AnsiChar 类型可以存储 256 个不同的 Unicode 字符; WideChar 则使用两个字节来表示一个字符。Char 属于通用类型, 目前等价于 AnsiChar, Delphi 将来可能使之扩展为 WideChar。

在 Pascal 语言中, 字符的表示是用单引 “'” 号括起来, 如 'a' 是表示由字母 a 构成字符。

以下介绍与字符处理有密切关系的两个函数。

● Ord(c:char):Integer

Ord 函数用于获取字符的 ASCII 码 (十进制), 其作用类似于 VB 中的 Asc() 函数, 如 Ord('A') 返回 65、Ord('a') 返回 97、Ord('b') 返回 98 等。

● Chr(n:Byte):Char

该函数的作用与上面讲到的 Ord 函数的作用相反, 它是用于返回 ASCII 字符集中次序值为 n 的字符。观察下面的例子:

```
var
  str:string;
  x: integer;
  c:Char;
begin
  c:='a';
  x:=Ord(c);
  write('字符'+c+'的次序值为: '); writeln(x);
  x:=98;
  c:=Chr(x);
  write('次序值为'); write(x);   writeln('的字符是: '+c);
  readln(str); //等待读入, 让程序停在等待状态, 以观察运行结果
end.
```

上例中, 首先利用 Ord 函数把字符 a 的次序值 97 打印出来, 然后利用 Chr 函数把次序值为 98 的字符 b 打印出来。从中完全可以体会到这两个函数的区别。

(3) 布尔类型。布尔类型有四种, 即 Boolean、ByteBool、WordBool 和 LongBool。常用的是 Boolean, 其他的类型是为了兼容其他语言和操作系统而给出的, 其中 Boolean 和 ByteBool

为单字节类型, WordBool 为双字节类型, LongBool 则为 4 字节类型。

布尔类型的值有两个: FALSE 和 TRUE。当 ByteBool、WordBool 和 LongBool 类型的数值为 0 时,可以认为是 FALSE,而非 0 时则可认为是 TRUE。布尔类型主要是用在条件语句和关系运算中。

(4) 枚举类型。枚举类型是一种用户自定义的简单类型,定义时列出所有可能的值,组成一组有序的标识符序列,但序列长度不能超过 255。定义的语法格式如下:

```
type typeName = (v1, v2, ..., vn);
```

其中 type 是关键字, typeName 是定义的枚举变量的名称, v₁, v₂, ..., v_n 是用户根据程序设计需要而定义的标识符序列。例如:

```
type day = (sun, mon, tue, wed, thu, fri, sat);
```

该语句的作用是把一周中的各天定义成为一个枚举类型,类型名为 day。这样,凡是类型为 day 的变量,它的取值范围只能为 {sun, mon, tue, wed, thu, fri, sat}, 即取值为一周中的某一天。

枚举类型通常与 case 语句连用,以达到某种控制目的。在下面的例子中,它是一个完整的控制台应用程序。

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
  type day = (sun, mon, tue, wed, thu, fri, sat); //注意:不是“:=,”而是“=”。
  var
    str:string;
    thisDay :day;
begin
  for thisDay := sun to sat do
    case thisDay of
      sun:   writeln('今天是星期天。');
      mon:   writeln('今天是星期一。');
      tue:   writeln('今天是星期二。');
      wed:   writeln('今天是星期三。');
      thu:   writeln('今天是星期四。');
      fri:   writeln('今天是星期五。');
      sat:   writeln('今天是星期六。');
    end;
  readln(str);
end.
```

枚举类型变量不能通过赋值语句或者 read 语句进行赋值,也不能通过 write 语句打印。例如,下列语句都是错误的:

```
thisDay := sun;
writeln(thisDay);
readln(thisDay);
```

(5) 子界类型。枚举类型定义时,要把所有的值一一罗列出来。当然,对数值个数比较小的枚举类型来说,这是件容易做得到的事情,但是当枚举值比较多时,如 1,2,...,一直到 100,要把所有的值全部列出来,那可是一件费时的的工作,而且容易出错。为此,在 Pascal 语言中给出了子界数据类型。

实际上,子界类型是整型、字符型、枚举型等某个有序类型的子集,是由它们当中的某两个常量来指定的该类型的值域,这两个常量就叫做子界类型的上界和下界。子界类型定义的语法格式是:

```
type typeName = 下界..上界;
```

例如:

```
type
```

```
    result = 0..100;
```

```
    letters = 'a'..'z';
```

子界类型的上界和下界必须是同一类型,而且必须是有序类型,如整型、字符型、枚举型、布尔型等;同时子界类型变量具有其所属基类类型的所有运算特征。

2. 实数类型

实数类型是指能存储具有小数部分的一种数据类型。在 Object Pascal 中,实数类型包括 single、real、double、extended、comp、currency 等六种类型,其特征如表 3.6 所示。

表 3.6 实数类型的基本特征

类型	数值范围	有效位数	字节数
single	$1.5 \times 10^{-45} \sim 7.4 \times 10^{38}$	7~8	4
real48	$2.9 \times 10^{-39} \sim 1.7 \times 10^{38}$	11~12	6
double (real)	$5.0 \times 10^{-324} \sim 1.7 \times 10^{38}$	15~16	8
extended	$7.4 \times 10^{-4932} \sim 1.1 \times 10^{4932}$	19~20	10
comp	$-2^{63}+1 \sim 2^{63}-1$	19~20	8
currency	$-922337203685477.5808 \sim 922337203685477.5807$	19~20	8

这些类型的补充说明如下。

- real48 类型是为了兼容以前的版本,它是早期版本中的 real 类型(如果要编译早期版本含 real 类型的代码,则需要把 real 转化为 real48,或者在编译时加入编译指令:{\$REALCOMPATIBILITY ON})。它的运行速度慢,空间消耗大。因此,除非在不得已的情况下,建议不要使用这种类型。
- double 同等于 real,是用得比较多的一种实数类型。
- extended 类型则提供了比 double 更高的精度,而且开销小,但是其兼容性不好。
- comp 类型的数据范围是 $-2^{63}+1 \sim 2^{63}-1$,比较大,常用于科学计算。
- currency 类型的最大优点是具有较高的精确度,兼容所有其他的实数类型,常用于财经计算。

实数可以用科学计数法表示,但是计数式中的各个部分要齐全,否则会产生错误,如 3.14E8、3.14E-8 都是正确的(它们分别表示 3.14×10^8 和 3.14×10^{-8}),但是 E8 和 3.14E 的写法都是错误的。

3.3.2 字符串类型

在声明字符串变量时常常用到关键字 string,它实际上是三种字符串类型的统称,这三种

类型是 ShortString（短字符串类型）、AnsiString（长字符串类型）和 WideString（宽字符串类型），它们的特征见表 3.7。

表 3.7 Object Pascal 中字符串类型的特征描述

类型	长度	存储空间
ShortString	255	2~256B
AnsiString(LongString)	2^{31}	4B~2GB
WideString	2^{30}	4B~2GB

1. 短字符串类型（ShortString）

短字符串类型的最大长度为 255，它是 Pascal 风格，与早期的 Pascal 版本兼容。

短字符串声明的语法格式为：

```
var strName: ShortString;
```

另外，还可以用 string 来声明短字符串，其格式如下：

```
var strName: String[n];
```

n 表示字符串的最大长度，当 n 为 255 时，该语句与上面声明的语句是等价的。

下面介绍与字符串有关的两个操作：

（1）访问字符串中的字符。访问的方法是通过字符串变量名加上字符串索引号来实现。

例如，对于字符串：

```
str := 'abcdefg';
```

如果要访问串中的第一个字符'a'，则可用下列方法：

```
c := str[1]; // c 为 Char 类型
```

类似地，str[2]、str[3]分别可以访问串中的第二个字符'b'和第三个字符'c'。

在索引号为 0 的地方存放的是字符串的实际长度，真正字符串的存储位置是从索引号为 1 的地方开始的。

（2）求取字符串的长度。求取字符串长度的方法很简单，只要调用 length 函数即可。例如，对于字符串：

```
str := 'abcdefg';
```

其长度为 length(str)。

2. 长字符串类型（AnsiString）

长字符串是 String 默认的类型，在理论上其最大长度为 2GB。该类型变量的存储空间是动态分配的，变量以 NULL 结尾。

长字符串变量声明的语法格式如下：

```
var strName: AnsiString;
```

等价于

```
var strName: String; //这是编程中最常用的方法
```

长字符串类型具有很多优点，能与 Windows 的多数函数例程兼容等，在编程中应多采用这种类型。

3. 宽字符串类型（WideString）

宽字符串类型与长字符串类型基本类似，也是采用动态分配内存的存储技术。不同之处

在于，宽字符串类型的数组元素是 `WideChar` 类型，而长字符串类型的数组元素是 `AnsiChar` 类型。长字符串类型的这种特性，使它能够处理国际字符集 `UNICODE` 等，有利于开发 `ActiveX` 应用程序。

在程序设计中通常使用 `String` 来定义字符串，例如

```
var str: String;
```

那么 `str` 是 `ShortString` 类型还是 `AnsiString` 类型呢？这要由编译指令来决定。如果使用 `{SH-}`，则 `str` 被认为属于 `ShortString` 类型；如果使用 `{SH+}`，则 `str` 被认为属于 `AnsiString` 类型。

3.3.3 指针类型

指针变量实际上是内存地址的变量，通过改变该变量值可以实现对不同内存单元的访问，这样使得程序设计变得更为灵活和方便。

指针类型变量定义的语法格式如下：

```
type PoinTyName = ^datatype;
```

其中，`PoinTyName` 为被定义的指针类型名；`datatype` 为用于定义的数据类型名。指针类型定义完成以后就可以用它来定义指针变量：

```
var p: PoinTyName;
```

其中，`p` 为被定义的指针变量。

下面通过一个具体的例子来理解指针变量的定义和运用。

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type IntPtr = ^Integer;
var
  str:string;
  A,B,C:Integer;
  p:IntPtr;
begin
  A:=100;
  B:=50;
  write('A,B 交换前,A='); write(A);   write(',B=');   writeln(B);
  p:=@A;
  C:=p^;
  p^:=B;
  B:=C;
  write('A,B 交换后,A='); write(A);   write(',B=');   writeln(B);
  readln(str);
end.
```

以上程序中，首先把 `IntPtr` 定义为整型指针类型，并用该类型定义指针变量 `p`；然后把整型类型变量 `A` 的地址赋给指针变量 `p`，这样在往下的程序代码中，`p^` 都是代表 `A` 这个单元，即对 `p^` 的操作和访问相当于对 `A` 的操作和访问，从图 3.4 所示的运行结果中可以证实这一点。

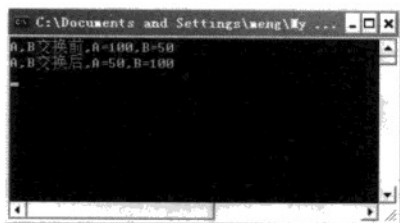


图 3.4 运行结果

从以上容易看到, 程序先在 `uses` 部分把 `^Integer` 定义成 `IntPtreter`, 然后在 `var` 部分就可以用 `IntPtreter` 来定义变量。这两条语句可合成一条语句: “`p: ^Integer;`”, 并把它放在 `var` 部分, 其作用是完全一致的。

3.3.4 结构类型

1. 数组类型

在 Object Pascal 中, 有两种类型的数组, 即静态数组和动态数组。

(1) 静态数组。静态数组类型变量定义的语法格式为:

```
var ArrayName: array[n..m] of DataType;
```

其中, `ArrayName` 为数组名; `DataType` 为用于定义的数据类型名称; `n, m` 为整数。例如, 定义长度为 200 的字符串数组类型变量可以用下面的语句:

```
var StrArray: array[1..200] of string;
```

访问或操作数组中某一个元素的方法是: 在数组变量名后面紧跟方括号, 方括号中标写元素的下标值。例如, 把数组 `StrArray` 中第 20 元素的值改为“张三丰”, 则可用下列语句:

```
StrArray[20] := '张三丰';
```

除了一维数组外, 还可以定义多维数组。以下是二维数组定义的语法格式:

```
var ArrayName: array[n1..m1, n2..m2] of DataType;
```

现定义二维数组:

```
twoArray: array[1..100, 1..10] of string;
```

多维数组的访问需要多个下标, 以下是访问二维数组 `twoArray` 的一条语句:

```
twoArray[1, 4] := '张三丰';
```

依此类推, 可以得到更高维数组的定义和访问方法。

(2) 动态数组。一维动态数组定义的语法格式如下:

```
var ArrayName: array of DataType;
```

可以看出, 与静态数组定义不同的是, 动态数组定义的格式中没有包含数组的下标。实际上, 在定义动态数组时, 并没有为数组分配内存, 要使数组具有数据存储能力还必须调用 `SetLength` 函数。`SetLength` 函数的作用是为指定的数组分配内存空间, 其语法格式为:

```
SetLength(ArrayName, ArrayLength);
```

其中, `ArrayName` 为已定义的数组名; `ArrayLength` 为正整数, 其作用就是为数组 `ArrayName` 分配 `ArrayLength` 个数据类型为 `DataType` 的内存单元。并且, 数组的索引号总是从 0 开始, 一直到 `ArrayLength-1`。如果多次对同一数组使用 `SetLength` 函数, 那么每一次调用都会对数组的内存重新进行分配。但是, 调用 `SetLength` 函数不会造成数据的丢失。下面的实例给出了验证。

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  str:string;
  Array1: array of integer;
begin

```

```

  SetLength(Array1, 3); //第一次对数组 Array1 分配存储空间

```

```

  Array1[0] := 100;

```

```

  Array1[1] := 200;

```

```

  Array1[2] := 300;

```

```

  writeln('数组 Array1 中的数据: ');

```

```

  writeln(Array1[0]);writeln(Array1[1]);writeln(Array1[2]);

```

```

  writeln("");

```

```

  SetLength(Array1, 4); //第二次对数组 Array1 分配存储空间 (变大了)

```

```

  writeln('调用 SetLength(Array1, 4)函数后, 数组 Array1 中的数据: ');

```

```

  writeln(Array1[0]);writeln(Array1[1]);writeln(Array1[2]);writeln(Array1[3]);

```

```

  writeln("");

```

```

  SetLength(Array1, 1);//第三次对数组 Array1 分配存储空间 (比上两次都小)

```

```

  writeln('调用 SetLength(Array1, 1)函数后, 数组 Array1 中的数据: ');

```

```

  writeln(Array1[0]);writeln(Array1[1]);writeln(Array1[2]);

```

```

  writeln("");

```

```

  readln(str);

```

```

end.

```

上面是一个完整的控制台应用程序，其运行结果如图 3.5 所示。

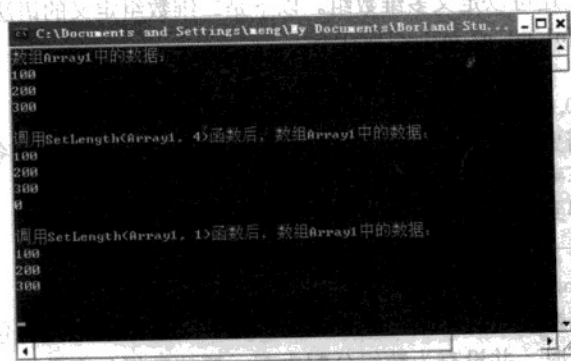


图 3.5 控制台应用程序的运行结果

(3) 与数组有关的几个函数。与数组有关的函数包括 Low、High、Length 等，其中，函数 Low 和 High 的作用是返回数组第一个索引的最小值和最大值，Length 则返回数组的长度，如果是多维数组则返回第一维的元素个数。

例如，如果定义：

```
var MyArray: array[-4..10] of integer;
```

则

Low(MyArray) = -4, High(MyArray) = 10, Length(MyArray) = 15。

2. 集合类型

集合类型是有序类型的子集，即是由有序类型的一些数值组成，而不能是实型或者其他构造类型。其定义的语法格式如下：

```
var SetName: set of baseType;
```

例如，下面是定义集合变量及其访问的例子。

```
var
    MySet1: set of 1..100;
    MySet2: set of Byte;
    MySet3: set of (sun, mon, tue, wed, thu, fri, sat);
    MySet4: set of Char;
    MySet5: set of 'a'..'z';
begin
    MySet1 := [1, 88, 90];
    MySet2 := [0, 255];
    MySet3 := [sun, fri];
    MySet4 := ['a', '9'];
    MySet5 := ['a', 'b'];
end.
```

从这个例子可以看出，集合类型变量的值是相应数据类型的数值的一个子集（整体），而不像枚举类型或子界类型那样，只是单个的元素。

3. 记录类型

记录类型是由多种不同数据类型的元素构造的，这些元素称为字段，每个字段由字段标识符和字段类型组成。记录类型定义的语法格式如下：

```
type
    RecordName = record
        field1: DataType1;
        ...
        field2: DataType2;
    end;
```

以下记录是对个人信息的描述：

```
Type person = record
    id: integer;
    name: string;
    age: 1..150;
    tel: string[7];
    address: string;
end;
```

这个记录由 5 个字段组成，它们都有各自的标识符和数据类型。

记录定义完成后，利用它来定义记录变量。以下是用记录 person 来定义两个记录变量 ZhangSan 和 LiSi：

```
var ZhangSan, LiSi : person;
```

显然，访问记录实际就是访问记录的字段值，方法是：在字段名前加上符号“.”，再加上记录变量名。例如，要访问 ZhangSan 的个人信息，可用下列语句实现：

```
ZhangSan.id := 20050501;  
ZhangSan.name := '张三';  
ZhangSan.age := 18;  
ZhangSan.tel := '8888888';  
ZhangSan.address := '中关村 105';
```

以上每一个语句前面都有字符串“ZhangSan”，这使得代码显得重复，而且也加重了编码的工作量，还可能引起不必要的错误。Pascal 语言中提供了 with 语句，使得可以“免去”这种重复的代码。例如，对上面的语句可以改成下面的 with 语句：

```
with ZhangSan do  
begin  
    id := 20050501;  
    name := '张三';  
    age := 18;  
    tel := '8888888';  
    address := '中关村 105';  
end;
```

3.3.5 过程类型

过程类型是利用过程或函数来构造的。例如，下面分别定义了三个过程类型：

```
type  
    Proc1 = Procedure;  
    Proc2 = Procedure(var n, m: integer);  
    Proc3 = Function(var x: single): single;
```

以上声明了三个过程类型，第一个是把不带参数的过程定义为过程类型，第二个则把带两个参数的过程声明为过程类型，最后一个则把带一个参数的函数声明为过程类型。

3.3.6 可变类型

可变类型 (variant) 是一种功能强大的数据类型，Delphi 中引入该类型的目的是为了支持 OLE 自动化操作。它主要用在数据类型可能发生变化的情况下，包括在编译期间和运行期间。此外，可变类型还可以支持简单的数据类型，如实数型、整型、字符型、布尔型等。

可变类型变量定义的语法格式为：

```
var variName: variant;
```

variant 变量定义时被初始化为一个特殊的初始值，即 Unassigned，表示 variant 变量还未被赋值。

3.4 熟悉 Object Pascal 的基本运算

在 3.2.4 节中已经列出了 Object Pascal 语言中的运算符，本节将结合这些运算符进一步介绍 Object Pascal 的一些基本运算，为 Object Pascal 编程打下基础。

3.4.1 赋值运算

在其他语言中一般是用等号“=”来作赋值运算符，而在 Pascal 语言中赋值符号则为“:=”

(冒号+等号)。赋值运算实际上就是将一个数值通过赋值符号“:=”赋给一个变量的操作，其格式为：

```
x := y;
```

其中，x 为变量；y 为变量、表达式或者常量。如果赋值符号“:=”的右边是一个表达式，则 Pascal 首先计算表达式的值，并把值的类型转化为与 x 相同的类型后才赋给 x。

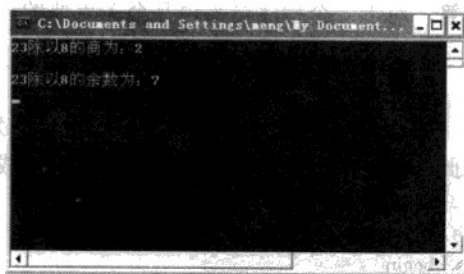
3.4.2 算术运算

算术运算符包括+（加）、-（减）、*（乘）、/（浮点数除法）、div（整数除法）、mod（整数取模）等。加、减、乘的算术运算与其他语言（如 C、VB 等）当中对应的运算是相同的，不同的是，在 Object Pascal 语言中，浮点数除法和整数除法运算用了不同的除法运算符，其中浮点数除法符号为“/”，而整数除法符号则为 div。注意下列代码的使用：

```
var
    n: integer;
    f: double;
begin
    n:= 18 / 8;    // 将产生编译错误，应该用符号 div
    f:=3.2 div 7.8; // 将产生编译错误，应该用符号/
end;
```

对于整数除法运算，其结果是返回两个整数相除后所得的商（整型），如果要取整数除法运算所得到的余数（整型），则应该应用运算符“mod”。从以下控制台应用程序的代码及其运行结果中可以充分地理解整数除法及运算符“/”、mod 及 div 的用法。

```
program Project1;
{$APPTYPE CONSOLE}
uses
    SysUtils;
var
    str:string;
    n,m: integer;
begin
    n:= 23 div 8;
    m:= 23 mod 8;
    write('23 除以 8 的商为: '); writeln(n);
    writeln("");
    write('23 除以 8 的余数为: '); writeln(m);
    readln(str);
end.
```



该程序运行结果如图 3.6 所示。

图 3.6 控制台应用程序的运行结果

3.4.3 逻辑运算与关系运算

逻辑运算操作的对象是布尔型数据，返回的结果也是布尔型数据。运算符包括 not（逻辑非）、and（逻辑与）、or（逻辑或）、xor（逻辑异或）等。假设 X 和 Y 分别为布尔型数据，则由它们及逻辑运算符构成的布尔表达式的真值表如表 3.8 所示。

表 3.8 逻辑表达式的真值表

X	Y	not X	X and Y	X or Y	X xor Y
F	F	T	F	F	F
F	T	T	F	T	T
T	F	F	F	T	T
T	T	F	T	T	F

表中, T 代表 true, F 代表 false。

由表 3.8 可知, not 是逻辑取反, 表达式 X and Y 为 True 当且仅当 X 和 Y 同时为 true, X or Y 为 false 当且仅当 X 和 Y 同时为 false, X xor Y 为 true 当且仅当 X 和 Y 的布尔值不同。

关系运算是两个算术表达式、字符(串)型的数据或布尔型数据进行比较, 结果为布尔类型。如果关系式成立, 则运算结果返回 true, 否则返回 false。表 3.9 给出了 Object Pascal 中关系运算符的功能说明。

表 3.9 关系运算符

运算符	功能说明
=	对于关系式 $X=Y$, 若 X 等于 Y, 则该式返回布尔值 true, 否则返回 false
\neq	对于关系式 $X \neq Y$, 若 X 不等于 Y, 则该式返回布尔值 true, 否则返回 false
\geq	对于关系式 $X \geq Y$, 若 X 大于或等于 Y, 则该式返回布尔值 true, 否则返回 false
\leq	对于关系式 $X \leq Y$, 若 X 小于或等于 Y, 则该式返回布尔值 true, 否则返回 false
>	对于关系式 $X > Y$, 若 X 大于 Y, 则该式返回布尔值 true, 否则返回 false
<	对于关系式 $X < Y$, 若 X 小于 Y, 则该式返回布尔值 true, 否则返回 false
in	对于关系式 $X \text{ in } Y$, 若 X 属于 Y, 则该式返回布尔值 true, 否则返回 false

关系运算的意义简单, 但在控制语句中有非常重要的应用, 所以正确地理解和运用关系运算, 对于一个程序员来说是至关重要的。

3.4.4 字符串运算

在 Object Pascal 语言中字符串运算符为 “+”, 它用于把两个字符串连接成一个字符串。因此, 字符串运算就是把两个字符串连接成一个字符串的操作。例如:

```
var
    str1, str2, str: string;
begin
    str1 := 'Chinese'; //把字符串'Chinese'赋给 str1
    str2 := 'man';     //把字符串'man'赋给 str2
    str := str1 + str2; //运算后, 得到字符串'Chinese man', 并把它赋给了 str
end;
```

3.4.5 指针运算和地址运算

指针运算是对指针类型变量进行操作或判断的过程。指针运算符包括 +、-、^、=、<, 其作用是:

- + 使指针变量增加一个偏移量。
- - 使指针变量减少一个偏移量。
- ^ 取指针所指向的单元的内容(值)。
- = 判断两个指针是否相等(指向同一个单元地址),相等则返回 true 否则返回 false。
- <> 判断两个指针是否不相等,不等则返回 true, 否则返回 false。

另外,符号“@”是单目运算符,用于获取变量的地址,得到的地址可以赋给指针变量。

例如,有下列代码:

```
var x: integer;
p: ^integer;
begin
    x:=99;
    p:=@x;
    write('x 的值为: '); writeln(x);
    write('p 指向的内存单元值为: '); writeln(p^);
end.
```

该代码段输出的结果如下:

x 的值为: 99

p 指向的内存单元值为: 99

可见,当把某一个变量 x 的地址赋给指针变量 p 以后,则该变量就可以利用指针 p 来访问,即 p^和 x 是等价的。

3.4.6 位运算

数据在内存中是以二进制的形式保存的,对二进制位的操作就形成了位运算。位运算是针对整数数据类型的存储位进行操作,包括按位对操作数进行移位和比较等。

位运算主要包括以下几种运算:按位取反(not)、按位与(and)、按位或(or)、按位异或(xor)、左移(shl)、右移(shr)等。这些运算符有的在形式上与逻辑运算符一样,但它们的操作数和返回结果是不同的:位运算的结果还是整数,而逻辑运算的结果则是布尔类型的。

下面分别对它们进行介绍(为说明起见,本小节中提到的 x 和 y 都是 byte 类型)。

1. 按位取反(not)

假如 byte 型变量 x 的存储形式为:

1	0	1	1	0	1	0	0
---	---	---	---	---	---	---	---

按位取反就是对每一位取反,即 1 变成 0, 0 变成 1。例如,在对 x 取反后,其存储形式就变为:

0	1	0	0	1	0	1	1
---	---	---	---	---	---	---	---

对 x 的按位取反运算记为: not x。例如:

x:=9; //00001001

y:=not x; //y 的二进制值为 11110110 (对应十进制 246)

2. 按位与 (and)

按位与是双目运算,它对两个变量的对应存储位进行符合下列规则的位运算:

&	0	1
0	0	0
1	0	1

假如 byte 型变量 y 的存储形式为:

1	0	1	0	1	1	0	1
---	---	---	---	---	---	---	---

则 x 和 y 在进行与运算后其结果 (的存储形式) 为:

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

对 x 和 y 的按位与运算记为: $x \text{ and } y$, 显然 $x \text{ and } y = y \text{ and } x$ 。

3. 按位或运算 (or)

按位或运算也是双目运算, 它对两个变量的对应存储位进行符合下列规则的位运算:

&	0	1
0	0	1
1	1	1

例如, 如果 x 与 y 在进行或运算以后, 其结果为:

1	0	1	1	1	1	0	1
---	---	---	---	---	---	---	---

对 x 和 y 的按位或运算记为: $x \text{ or } y$, 显然 $x \text{ or } y = y \text{ or } x$ 。

4. 按位异或运算 (xor)

按位异或运算也是双目运算, 它是对两个变量的对应存储位进行符合下列规则的位运算:

&	0	1
0	0	1
1	1	0

例如, 如果 x 与 y 在进行异或运算以后, 其结果为:

0	0	0	1	1	0	0	1
---	---	---	---	---	---	---	---

对 x 和 y 的按位异或运算记为: $x \text{ xor } y$, 显然 $x \text{ xor } y = y \text{ xor } x$ 。

5. 左移运算 (shl)

左移运算是单目运算, 它将变量的二进制位左移既定的若干位, 低位补 0, 高位左移出 (弃用)。例如, 对变量 x 进行 3 位左移的示意如图 3.7 所示。

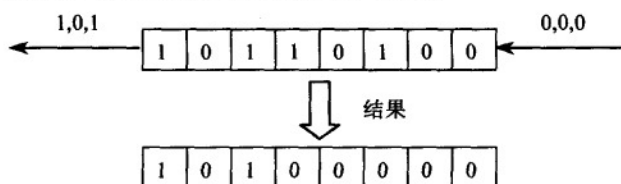


图 3.7 对变量 x 进行 3 位左移的结果

对 x 的左移 n 位运算记为: $x \text{ shl } n$ 。左移一位相当于原值乘以 2, 所以左移 3 位则相当于 x 乘以 2^3 , 即 $(x \text{ shl } 3) = x * 2^3$ 。例如:

```
x:=9;           //00001001
y:=(x shl 3);    //y 的二进制值为 01001000 (对应十进制 72)
```

6. 右移运算 (shr)

右移运算则是把二进制位右移既定的若干位, 低位移出后不用, 高位则均“填”以原值的最高位。所以, 每一位就相当于对原值除以 2。如果对 x 左移 3, 则相当于对 x 除以 2^3 , 即 $(x \text{ shr } 3) = x \text{ div } 2^3$ 。

3.4.7 运算符的优先级

一般来说, 圆括号的优先级最高, 然后是算术运算符, 最后是位运算符和赋值运算符。图 3.8 表示了运算符的优先级关系, 其中同一层的则表示优先级一样, 运算时将以先后关系来决定。

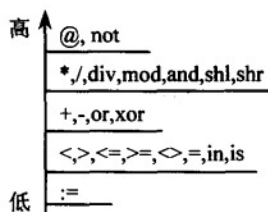


图 3.8 Object Pascal 运算符的优先级

3.5 熟悉 Object Pascal 的流程控制

有效的流程控制是程序能够完成既定功能的基本保证。与 C、C++、VB 类似, Object Pascal 的流程控制主要由流程控制语句来实现, 主要包含以下几种:

- if 语句。
- case 语句。
- while 语句。
- for 语句。
- repeat 语句。
- 转移语句: goto, break, label, continue, return。

3.5.1 if 语句

在程序中, 有一些语句或复合语句的执行是有条件的, 也有的时候需要在多个语句或复合语句之间的执行作出选择。这时需要一些选择语句才能完成, if 就是一种最基本的、最常用的选择语句。

在 Object Pascal 中 if 语句有三种形式: if ...then...、if ... else ...和 if ... else if ... else ...。

1. if ...then...句型

这种 if 语句的语法规则如下:

if 布尔表达式 then 语句 (或者复合语句);

如果布尔表达式的返回值为 `true`，则执行后面的语句（或者复合语句），否则什么都不执行，其逻辑流程见图 3.9。

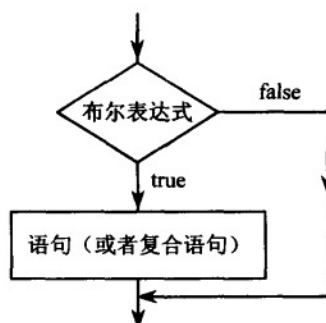


图 3.9 if ...then...句型的逻辑流程图

在 Pascal 中，复合语句是指由若干个有序的语句组成，每个语句之间用分号隔开，并用 `begin` 和 `end` 将这些语句括起来。另外，`begin` 和 `end` 不是语句，仅起语句括号作用，分号也不是语句的组成部分，只起分隔语句的作用。例如：

```

var
  x,y,z:integer;
begin
  x:=1;
  y:=2;
  if x<y then z:=y; //执行一条语句
  if x>y then      //执行一个复合语句
  begin
    z:=x;
    x:=y;
    y:=z;
  end;
end.
  
```

} 复合语句

2. if ... else ... 句型

有时要从两个语句或者复合语句中选择一项继续执行程序，这时要用到 `if ... else ...` 句型的 `if` 语句。

`if ... else ...` 句型的语法格式如下：

```

if 布尔表达式
  语句 1 (或者复合语句 1);
else
  语句 2 (或者复合语句 2);
  
```

该句型中，如果布尔表达式返回 `true` 则执行语句 1（或者复合语句 1），否则（布尔表达式返回 `false`）执行语句 2（或者复合语句 2）。

在这种 `if` 语句句型中，不管布尔表达式的返回值如何，总是有语句（或者复合语句）被执行，其逻辑流程见图 3.10。

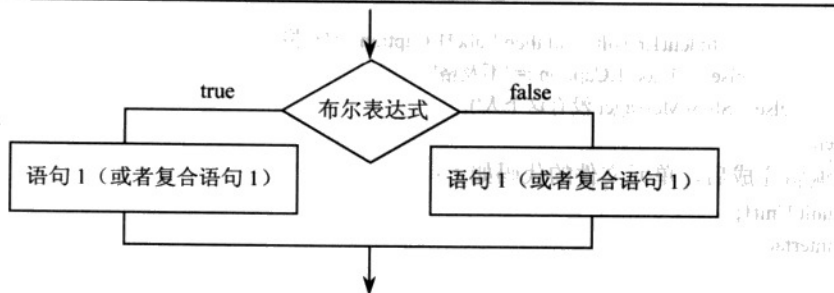


图 3.10 if...else...句型的逻辑流程图

现在用 if ... else ... 句型来实现学生成绩查询的程序，查询方法是：输入学生的姓名即可查询该学生成绩是否及格。

(1) 首先创建一个 VCL 应用程序，然后在窗体上创建 TLabel 对象、TButton 对象和 TEdit 对象，并做一些必要的属性设置（如字体、大小等），结果如图 3.11 所示，并把工程命名为 ResultQuery。

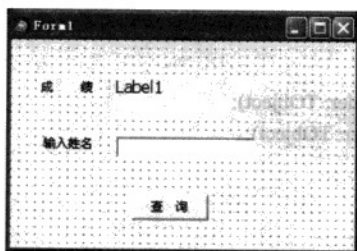


图 3.11 查询界面

(2) 在代码编辑器中打开单元文件，在 interface 部分定义学生记录，代码如下：

```

type stu = record
    name:string;
    result:single;
end;


```

然后定义类型为 stu 的记录变量 student1：

```

var student1:stu;

```


(3) 利用工具栏上的快捷按钮 ，转入窗体设计器，双击窗体编辑处进入 TForm1.FormCreate 函数代码编辑的地方，输入下列代码：

```

student1.name:='张无忌';
student1.result:=98;

```

即设置 student1 变量的初始值。

(4) 再利用工具栏上的快捷按钮 ，转入窗体设计器，双击 Button 按钮进入 TForm1.Button1Click 函数代码编辑的地方，把 Button1Click 修改成下列形式：

```

procedure TForm1.Button1Click(Sender: TObject);
var str:string;
begin
    str:=Edit1.Text;
    if str='张无忌' then

```



```

        if student1.result>=60 then Label1.Caption := '及格'
        else Label1.Caption := '不及格'
    else ShowMessage('没有这个人');
end;

```

编辑完成后，单元文件的代码如下：

```

unit Unit1;
interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;

type
    TForm1 = class(TForm)
        Label1: TLabel;
        Button1: TButton;
        Label2: TLabel;
        Label3: TLabel;
        Edit1: TEdit;
        procedure Button1Click(Sender: TObject);
        procedure FormCreate(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;
    stu = record
        name:string;
        result:single;
    end;

var
    Form1: TForm1;
    student1:stu;
implementation
    {$R *.dfm}
    procedure TForm1.FormCreate(Sender: TObject);
    begin
        student1.name:='张无忌';
        student1.result:=98;
    end;

    procedure TForm1.Button1Click(Sender: TObject);
    var str:string;
    begin
        str:=Edit1.Text;

```

```

if str='张无忌' then
    if student1.result>=60 then Label1.Caption := '及格'
    else
    begin
        Label1.Caption := '不及格';
    end
else ShowMessage('没有这个人');
end;
end.

```

} 复合语句

在上述代码的 Button1Click 函数中用了两个 if...else...语句，它们是嵌套关系。外层的 if 语句是在两个语句之间的执行作出选择，内层的 if 语句则在一个语句和一个复合语句（是由一条语句构成）之间的执行作出选择。图 3.12 所示为工程 ResultQuery 运行的一个结果。

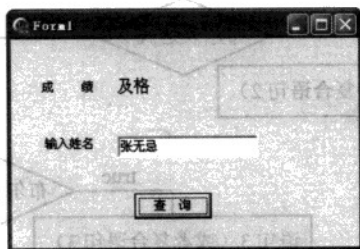


图 3.12 工程 ResultQuery 运行的界面

显然，这个程序几乎不具有应用功能，但在下文中将以此为列，逐步完善它，最终将使它具备一定应用功能，同时在这一过程中讲解相关语句的使用方法。

在 if...else...句型的 if 语句中，紧跟“else”的前一行代码的结束不能有分号“;”，否则将产生编译错误。例如，下面的两个 if 语句都是错误的：

```

if str='张无忌' then
    if student1.result>=60 then Label1.Caption := '及格'; ← 错误
    else
    begin
        Label1.Caption := '不及格';
    end; ← 错误
else ShowMessage('没有这个人');

```

3. if ... else if ...else...句型

当要作多次判断时，则要用到 if ... else if ...else...句型。该句型的语法规则如下：

```

if 布尔表达式 1 then
    语句 1（或者复合语句 1）；
else if 布尔表达式 2 then
    语句 2（或者复合语句 2）；
...
else if 布尔表达式 n then
    语句 n（或者复合语句 n）；
else
    语句 n+1（或者复合语句 n+1）；

```

在该句型中, 如果布尔表达式 1 的返回值为 true, 则执行语句 1 (或者复合语句 1), 否则计算布尔表达式 2; 如果布尔表达式 2 返回 true, 则执行语句 2 (或者复合语句 2), …… , 如果布尔表达式 n 返回 true, 则执行语句 n (或者复合语句 n), 否则执行语句 $n+1$ (或者复合语句 $n+1$), 即如果所有的布尔表达式 (布尔表达式 1 至布尔表达式 n) 都为假 (均返回 false), 则返回执行语句 $n+1$ (或者复合语句 $n+1$)。

在这种句型中, 也总是有一个语句 (或复合语句) 被执行, 当 $n=3$ 时其逻辑流程图见图 3.13。

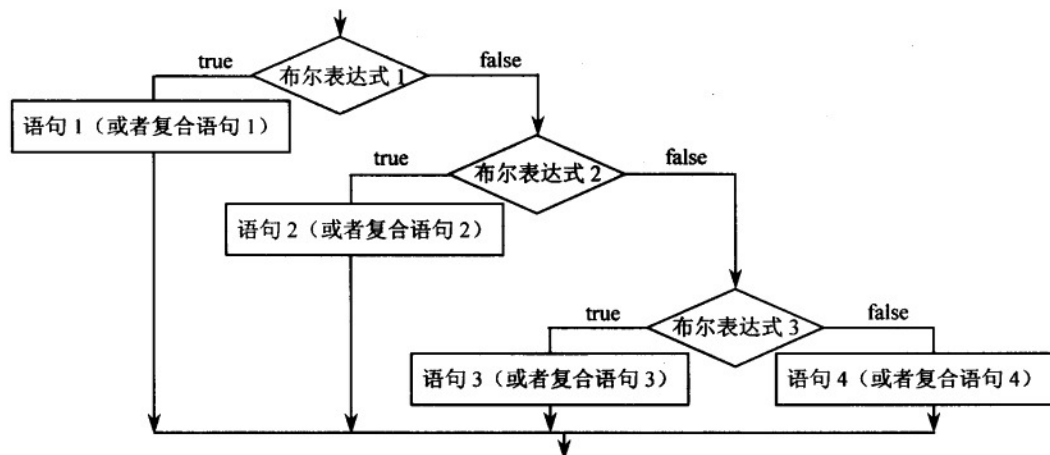


图 3.13 if ... else if ... else ... 句型的逻辑流程图

在 ResultQuery 程序中, 实际上成绩还可以进一步细分, 即及格当中还可以分为: 优秀 (90~100)、良好 (80~89)、中等 (70~79)、及格 (60~69)。这样, 连 “不及格 (<60)” 在一起, 一共有五个等级, 学生查成绩的时候应该能给出成绩的等级信息。为此, 对处理 Button1 按钮 Click 事件的函数 Button1Click 进行修改, 其结果代码如下:

```

procedure TForm1.Button1Click(Sender: TObject);
var
  str:string;
  x:double;
begin
  str:=Edit1.Text;
  if str<>'张无忌' then
  begin
    Label1.Caption := "";
    ShowMessage("没有这个人");
    exit; //退出函数
  end;
  x:=student1.result; //取成绩
  if x>=90 then //if ... else if ... else ... 句型的运用
    Label1.Caption := '优秀'
  else if x>=80 then

```

```

Label1.Caption := '良好'
else if x >= 70 then
    Label1.Caption := '中等'
else if x >= 60 then
    Label1.Caption := '及格'
else
    Label1.Caption := '不及格';
end;
end.

```

代码编写好后运行该程序，然后在编辑框中输入“张无忌”，结果如图 3.14 所示。

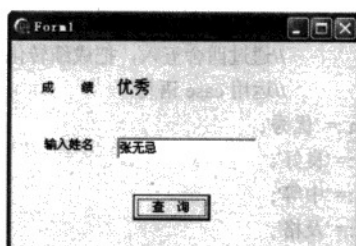


图 3.14 张无忌的成绩查询结果

3.5.2 case 语句

if 语句每次判断后可以并最多能作出两种选择，但有时要作出两种以上的选择，这时就要用到 case 语句。case 语句相当于 C/C++ 中的 switch 语句，其语法格式为：

case 表达式 of

值表 1: 语句 1 (或复合语句 1);

值表 2: 语句 2 (或复合语句 2);

... ..

值表 n: 语句 n (或复合语句 n);

[
else

语句 n+1 (或复合语句 n+1);

]

end;

case 语句的基本原理是：首先计算表示式的值，然后把得到的值与值表 1, ..., 值表 n 进行匹配，如果匹配成功则执行其后的语句或复合语句，执行完后就跳出 case 语句。其中，else 部分是可选的（中括号表示可选），它表示，如果所有的匹配都不成功则执行 else 后面的语句或复合语句。

对于工程 ResultQuery，也可以用 case 语句来实现学生不同等级成绩的查询，只要修改 Button1Click 函数中的部分代码即可，其代码如下：

```

procedure TForm1.Button1Click(Sender: TObject);
var
    str:string;
    x:double;

```

```

n:integer;

begin
  str:=Edit1.Text;
  if str<>'张无忌' then
  begin
    Label1.Caption := '';
    ShowMessage('没有这个人');
    exit;    //退出函数
  end;
  x:=student1.result;           //读取成绩（实数型）
  n:=round(x);                 //通过四舍五入，把成绩转化为整型
  case n of                    //运用 case 语句
    90..100: Label1.Caption := '优秀';
    80..89: Label1.Caption := '良好';
    70..79: Label1.Caption := '中等';
    60..69: Label1.Caption := '及格';
    0..59: Label1.Caption := '不及格';
  else
    Label1.Caption := '非法分数';
  end;
end.

```

(1) 在 Object Pascal 中，对于任意的两个值表 i 和值表 j ，彼此不能相同，否则会出现编译错误。

(2) 表达式的值必须是有序类型，而像浮点型、字符串型就不能作为 case 的表达式。因此，在上面的程序中要把 double 型的成绩转化为整型，然后再运用 case 语句。

(3) 值表类型与表达式的类型必须一致。

3.5.3 for 语句

for 语句是一种循环语句，是编程中用得最多的一种语句之一。其一般语法格式为：

```

for 控制变量:=初值 to 终值 do
  循环体;

```

for 语句的执行过程为：对控制变量赋初值，每次在执行循环体之前首先将当前控制变量的值与终值对比，如果小于或等于终值则执行一次循环体中的语句，并将控制变量当前值的后继值赋给控制变量，直到控制变量的值大于终值才停止执行循环体并跳出来。图 3.15 显示了 for 语句的逻辑流程图。

对于上一节的工程 ResultQuery，注意到其中只保存了一个学生的信息，这样显然没有查询意义。为此考虑多个学生的情况。为简单起见，程序用记录型数组来保存多个学生的成绩信息（而不用数据库）。因此，把 student1 定义为数组类型：

```

student1: array[1..200] of stu; //考虑 200 个学生

```

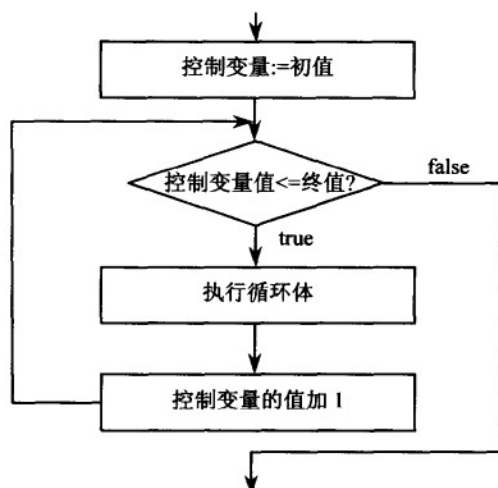


图 3.15 for 语句的逻辑流程图

然后在 FormCreate 函数中对数组 student1 进行初始化，代码如下：

```

procedure TForm1.FormCreate(Sender: TObject);
begin
  student1[1].name:='张三丰'; student1[1].result:=100;
  student1[2].name:='张无忌'; student1[2].result:=97;
  student1[3].name:='宋远桥'; student1[3].result:=88;
  student1[4].name:='俞莲舟'; student1[4].result:=85;
  student1[5].name:='俞岱岩'; student1[5].result:=79;
  student1[6].name:='张松溪'; student1[6].result:=77;
  student1[7].name:='张翠山'; student1[7].result:=79;
  student1[8].name:='殷梨亭'; student1[8].result:=68;
  student1[9].name:='莫声谷'; student1[9].result:=69;
  student1[10].name:='宋青书'; student1[10].result:=45;
end;

```

之后，用 for 语句对输入的名称进行匹配，如果匹配成功则读出该学生的成绩，代码如下：

```

for i:=1 to 10 do
begin
  if str=student1[i].name then
  begin
    x:=student1[i].result;      //读出学生成绩
    break;                     //退出 for 循环
  end;
end;

```

实现输出成绩所属的等级信息沿用 case 语句，这样单元的所有代码如下：

```

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

```

```

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Button1: TButton;
    Label2: TLabel;
    Label3: TLabel;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  stu = record
    name:string;
    result:single;
  end;
var
  Form1: TForm1;
  student1: array[1..10] of stu;    //原来是 student1:stu;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
  student1[1].name:='张三丰'; student1[1].result:=100;
  student1[2].name:='张无忌'; student1[2].result:=97;
  student1[3].name:='宋远桥'; student1[3].result:=88;
  student1[4].name:='俞莲舟'; student1[4].result:=85;
  student1[5].name:='俞岱岩'; student1[5].result:=79;
  student1[6].name:='张松溪'; student1[6].result:=77;
  student1[7].name:='张翠山'; student1[7].result:=79;
  student1[8].name:='殷梨亭'; student1[8].result:=68;
  student1[9].name:='莫声谷'; student1[9].result:=69;
  student1[10].name:='宋青书'; student1[10].result:=45;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  str:string;
  x:double;
  n,i:integer;
begin
  str:=Edit1.Text;
  for i:=1 to 10 do
  begin
    if str=student1[i].name then

```



```

begin
    x:=student1[i].result;
    break;           //退出 for 循环
end;
end;
if i>10 then begin ShowMessage('没有这个人'); exit; end; //退出函数
n:=round(x);        //通过四舍五入, 把成绩转化为整型
case n of           //运用 case 语句
    90..100: Label1.Caption := '优秀';
    80..89: Label1.Caption := '良好';
    70..79: Label1.Caption := '中等';
    60..69: Label1.Caption := '及格';
    0..59: Label1.Caption := '不及格';
else
    Label1.Caption := '非法分数';
end;
end;
end.

```

这样, 通过利用 for 语句, 就可以实现任意学生的成绩信息查询功能。例如, 运行程序后输入“宋青书”, 其结果如图 3.16 所示。

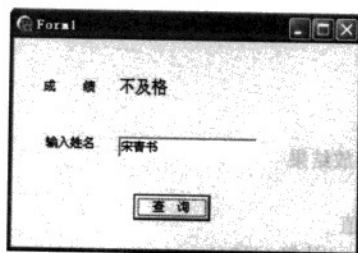


图 3.16 宋青书的成绩查询结果

另外, for 语句还有一种类型, 其语法格式如下:

```

for 控制变量:=初值 downto 终值 do
    循环体;

```

在这种类型中, 初值“大于”终值, 其执行的过程为: 对控制变量赋初值, 每次在执行循环体之前首先将当前控制变量的值与终值对比, 如果大于或等于终值, 则执行一次循环体中的语句并将控制变量当前值的前驱值赋给控制变量, 直到控制变量的值小于终值才停止执行循环体并跳出来。

3.5.4 while 语句

while 语句也是常用的循环语句, 其语法格式为:

```

while 布尔表达式 do
    循环体;

```

while 语句的执行过程为: 每次在执行循环体之前先判断当前布尔表达式的值是否为 true, 如果为 true 则执行一次循环体, 否则退出 while 语句, 执行 while 语句后面的语句。其流程示

意见图 3.17。

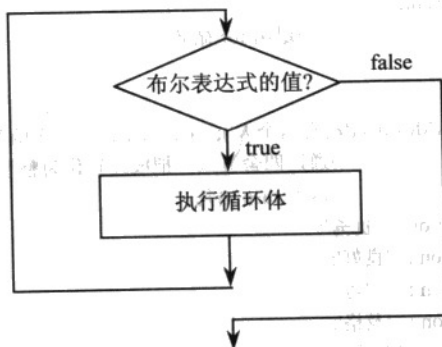


图 3.17 while 语句的逻辑流程图

以下程序采用 while 语句求 $1 \times 2 \times \dots \times n$ 的乘积，其中 n 是由用户输入的整数。它是个控制台应用程序，以下是完整的代码：

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  str:string;
  i,sum:integer;
begin
  sum:=1; //初始化，用于存放结果
  i:=1;
  readln(str); //读入 n 的值
  while i<=StrToInt(str) do //计算  $1 \times 2 \times \dots \times n$ 
  begin
    sum:=sum*i;
    i:=i+1;
  end;
  writeln('1×2×...×'+str+'='+IntToStr(sum));
  readln(str); //让程序“停”下来，以观看结果
end.
```

该程序的运行结果如图 3.18 所示。

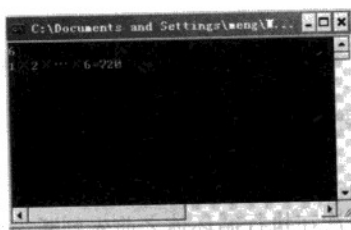


图 3.18 程序运行结果

3.5.5 repeat 语句

while 循环语句的特点是“先判断，后执行”，但有时候却需要“先执行，后判断”，以保证循环体至少被执行一次。repeat 语句正是具有这样功能的循环语句，它的一般语法格式为：

```
repeat  
    循环体;  
until 布尔表达式
```

repeat 语句执行的过程为：首先执行一次循环体，然后判断当前布尔表达式的值是否为 false，如果为 false 则再一次执行循环体，否则退出 repeat 语句，执行 repeat 语句后面的语句。其流程示意图 3.19。

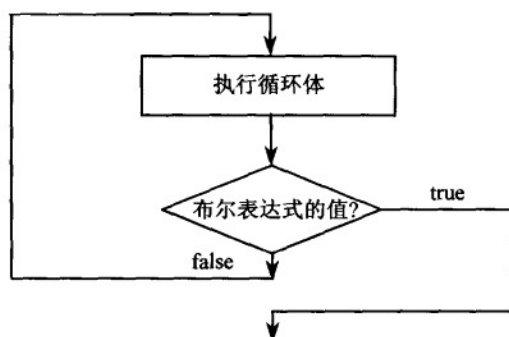


图 3.19 repeat 语句的逻辑流程图

把上面程序中的 while 语句改为 repeat 语句，同样可以计算 $1 \times 2 \times \cdots \times n$ 的值，其更改的代码如下：

```
repeat  
    sum:=sum*i;  
    i:=i+1;  
until i>StrToInt(str);
```

3.5.6 转移语句

转移语句是指 break 语句、continue 语句、exit 语句、goto 语句和 halt 语句等。break 语句和 continue 语句经常和循环语句搭配使用，这在前面已有例子。exit 语句则用于退出函数或代码块（复合语句），前面的例子也已经应用了。

halt 语句与 exit 语句有点类似，但 halt 是用于终止程序的运行（非正常），返回操作系统。在此重点介绍一下 goto 语句的使用方法。

goto 语句可以实现从一个地方无条件地跳转到另一个地方的功能。在 Object Pascal 语言中，goto 语句的语法格式为：

```
goto 标号;
```

标号可以是 0~9999 中的一个数，也可以是一个合法的标识符。标号在使用之前要声明，声明的语法格式为：

```
label 标号;
```

以下是使用 goto 语句的示例程序，该程序用于计算 $(1 \times 1) + (1 \times 2) + (1 \times 2 \times 3) + (1 \times 2 \times 3 \times 4) + \dots + (1 \times 2 \times \dots \times n)$ 的值。

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  str:string;
  i,n,sum,count:integer;
label  locAdd,1000; //分别把标号定义为标识符和数字

begin
  sum:=0; //初始化，用于存放结果
  count:=1;
  n:=1;
  readln(str); //读入 n 的值
  locAdd: //外循环，求和
    count:=1;
    i:=1;
    1000: //内循环，求积
      count:= count*i;
      i:=i+1;
      if i<=n then goto 1000;
    sum:=sum+count;
    n:=n+1;
  if n<=StrToInt(str) then goto locAdd;
  write(sum);
  readln(str); //让程序“停”下来，以观看结果
end.
```

从该程序中可以看出，goto 语句非常灵活，利用 goto 语句也可以实现循环语句的功能。同时也注意到，这个程序的可读性较差，很难看出它的层次性。多次使用该语句很容易增加程序出错的机会，特别是在程序代码很多的时候。建议在一般情况下不要使用 goto 语句。实际上，很多语言当中都去掉了 goto 语句。

3.6 熟悉 Delphi 的过程和函数

过程和函数是实现特定功能的程序段，它们都可以当作语句来调用，包括被其他程序调用和自身递归调用。两者的不同之处在于，函数有返回值，而过程没有返回值，所以函数可以作为表达式来运算，而过程则不行。

3.6.1 过程的定义与调用

1. 过程的定义

过程定义的语法格式如下：

```

procedure 过程名(参数列表)  ← 过程头部分
  过程变量声明  ← 过程声明部分
begin
  过程体
end;
  
```

} 过程体部分

可以看出，过程由三部分组成：过程头、过程声明和由 `begin` 和 `end` 括起来的过程体。其中，过程头部用于指定过程名和声明形式参数，过程名可以是任意一个合法的标识符，当然，过程名应该具有一定的意义，以便于程序修改和维护；参数可以没有，或者有一个或多个，多个参数用分号隔开。过程声明部分用于声明过程中用到的变量（局部变量），也可以没有过程声明部分（如果过程不需要用局部变量）。过程体由过程的语句序列组成，它们是完成过程既定功能的程序代码。

下面定义了过程 `exch`，它用于交换两个整型变量的值。

```

procedure exch(var n,m:integer);
var t:integer;
begin
  t:=n;
  n:=m;
  m:=t;
end;
  
```

2. 调用过程的方法

过程调用比较简单，其格式为：

过程名(实参);

其中，实参是相对过程定义的参数列表，即形参而言的，实参和形参要匹配。例如，调用过程 `exch` 的语句为：

```
exch(x,y);
```

其中，`x` 和 `y` 都是已经定义过的 `integer` 类型的变量。

在下面的控制台应用程序中，首先定义了过程 `exch`，然后对其进行调用。其代码如下：

```

program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var
  str:string;
  x,y:integer;
procedure exch(var n,m:integer); //定义过程 exch
var t:integer;
begin
  t:=n;
  n:=m;
  m:=t;
end;
begin
  x:=2; y:=9;
  writeln(IntToStr(x)+' '+IntToStr(y));
  
```

```
    exch(x,y);                //调用过程
    writeln(IntToStr(x)+';'+IntToStr(y));
    readln(str);              //让程序“停”下来，以观看结果
end.
```

运行后，其输出结果为：

```
2,9
9,2
```

可见，变量 x 和 y 的值已经被交换了。这个例子虽然简单，但是较好地给出了过程定义和调用的方法。

3.6.2 函数的定义与调用

函数有返回值而过程却没有，这使得函数的定义与过程的定义稍有区别。函数定义的语法格式为：

```
function 函数名(参数列表): 返回值类型;
    函数变量声明
begin
    函数体
end;
```

以下定义了一个求浮点型变量 x 的绝对值的函数：

```
function abs(var x:double):double;
begin
    if x<0 then abs:=-x
    else abs:=x;
end;
```

函数定义完了以后就可以调用了，可以用下列语句实现：

```
y:=abs(x);
```

由上面的例子，读者不但推出函数的基本使用方法。过程或函数的头部都是以分号结尾，即结尾的分号不能省略。

3.7 熟悉 Delphi 中的面向对象编程技术

Object Pascal 就是在 Borland 公司的 Pascal 语言的基础上，引入 OOP 技术（面向对象技术）而形成的一种语言。因此，面向对象的编程技术是 Object Pascal 的一大特点。这使得用 Delphi 开发的应用程序具有更好的可维护性和可重用性，以减少软件开发成本。本节将介绍 Delphi 中面向对象的编程技术。

3.7.1 熟悉对象与类

类是对若干相似对象的描述，是对现实世界对象的抽象。如果把人看做一个类，那么人这个类可能涉及的特征是姓名、性别、身高、籍贯等属性。而具体的一个人，比如说张三，就是一个对象。这样，通过人这一个类就可把所有的人——张三、王五、李四等这些具体对象的特征都抽象出来了，显然这种抽象方法为对复杂问题的建模、以至于最终的解决提供了有效的途径。

类是面向对象编程的基本单元，它是由关于对象的数据以及对于这些数据进行操作的函数（又称方法）所组成的。对象是在类被实例化后形成的，是由类来定义的，是对类的具体实现。

3.7.2 熟悉类的声明

在前面编写的 VCL 应用程序中已经接触过类。例如，在 2.3 节创建的工程 HelloWorldProject 中，打开单元文件 Unit4.pas，可以找到定义如下的类：

```
type
  TForm4 = class(TForm)
    LabHello: TLabel;
    Button1: TButton;
    Button2: TButton;
    procedure Button2Click(Sender: TObject);
    procedure btShowHello(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

从这个类的定义中，可以看出类有点类似记录类型，其与记录类型的不同之处仅在于：类除了包含字段定义以外，还包含方法的定义。因此，类是把数据和方法封装起来的一种“数据类型”。实际上，类正是由数据和方法构成的，数据和方法又通常称为成员变量和成员方法，其结构描述如图 3.20 所示。

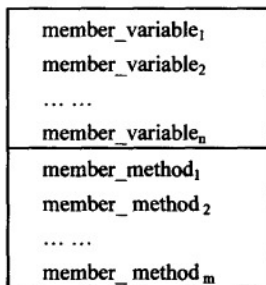


图 3.20 类的基本结构

因此，类的定义过程主要包括成员变量和成员方法的定义，此外，还需类说明关键字 class、类名及类属性。类名必须是合法的标识符，类属性是一些修饰类的关键字，包括 public 和 private 等。这样，类定义的语法格式可以描述为：

```
Type
  ClassName = Class(AncestorClass)
    定义成员变量 1;
    定义成员变量 2;
    ... ..
    定义成员变量 n;
    定义方法 1;
    定义方法 2;
```



```
... ..  
    定义方法 m;  
end;
```

其中, AncestorClass 表示基类, 如果不需要基类, 则把 Class 后面的圆括号去掉。

为进一步掌握 OOP 的编程技术和方法, 先创建一个 VCL 应用程序, 命名为 ClassProject。在代码编辑器中打开它的单元文件, 然后在 interface 部分声明类 TPerson, 代码如下:

```
TPerson = class  
    Private Name:string;  
    Private ID_Card:string;  
    Private Address:string;  
    Public Salary: currency;  
    Public procedure SetPerson(Name,ID_Card,Address:String; Salary:currency);  
end;
```

然后在 implementation 部分实现类中定义的方法 SetPerson, 其实现代码如下:

```
procedure TPerson.SetPerson(Name,ID_Card,Address:String; Salary:currency);  
begin  
    self.Name := Name;  
    self.ID_Card := ID_Card;  
    self.Address := Address;  
    self.Salary := Salary;  
end;
```

这样名为 TPerson 的类就被创建了。实际上, TPerson 类是一个关于人的类, 定义了 4 个成员变量。

- Name: 人名。
- ID_Card: 身份证号码。
- Address: 住址。
- Salary: 工资。

以及一个公有成员方法: procedure SetPerson(Name,ID_Card,Address:String; Salary:currency), 它用于对各成员变量进行设置。

另外, public 和 privated 的含义如下:

- public: 说明为公有成员变量。公有成员变量可以被类内、类外的所有方法访问, 其作用域最广。默认的访问模式是 public。
- privated: 说明为私有成员变量。私有成员变量仅被类内的方法所访问。

3.7.3 创建对象及对象成员的引用

类是一种抽象的“数据类型”, 它本身并不占用内存空间, 当它被实例化为对象才会“侵占”系统资源, 形成“实实在在的东西”。类实例化的过程就是对象创建的过程, 是为对象动态分配内存的过程。这个任务是由 Create 方法来完成, 这个方法是每一个 Class 都具有的隐含方法。

创建一个对象可以分为两步。

(1) 声明对象, 语法格式为:

```
var 对象名: 类名;
```

用上述语法声明后,对象名还只是一个指针,没有分配到内存。

(2) 调用构造函数 Create 来分配内存,语法格式为:

对象名:= 类名.Create;

例如,利用上面定义的类 TPerson 来创建一个对象 person,可由下面两条语句完成:

```
var person: TPerson;
```

```
person:=TPerson.Create;
```

创建了对象以后,就可以使用对象了,如给对象成员赋值、访问对象成员等。下面是一个定义类、创建对象和使用对象的一个完整的例子。

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  TPerson=class //定义类
    Name:string;
    ID_Card:string;
    Address:string;
    Salary: currency;
    procedure SetPerson(Name,ID_Card,Address:String; Salary:currency);
  end;

var
  str:string;
  person: TPerson; //声明对象

procedure TPerson.SetPerson(Name,ID_Card,Address:String; Salary:currency); //实现类的方法
begin
  self.Name := Name;
  self.ID_Card := ID_Card;
  self.Address := Address;
  self.Salary := Salary;
end;

begin
  person:=TPerson.Create; //创建对象
  person.SetPerson('黄蓉','1234567890','中南大学',2000.88);
  writeln('姓名: '+person.name);
  writeln('身份证号码: '+person.ID_Card);
  writeln('地址: '+person.Address);
  write('薪水: '); writeln(person.Salary);
  readln(str); //让程序“停”下来,以观看结果
end.
```

该程序运行结果如图 3.21 所示。


```
TPerson.free;
```

```
end;
```

这样，在程序退出时，就可以调用析构函数 Destroy 来撤销所创建的对象，方法是：

```
person.Destroy;
```

3.7.5 熟悉类的封装、继承和多态性

面向对象编程有三个主要特性，那就是类的封装、继承和多态性。

1. 类的封装性

在前面已经看到，类是由一系列的成员变量和成员方法组成，成员变量一般只由本类的成员方法来访问，而不为本类以外的其他类的成员方法所访问。这样，其他方法要访问类的成员变量，就必须通过本类的成员方法来实现。实际上，类就是把数据和方法封装起来而形成的一个结构，这就是类的封装性。

当然，类的封装性并不是死的，它在一定程度上还是可以被“打破”的。实际上，类封装的程度是由下列的关键字来决定的。

(1) Private (私有)。如果在成员变量前面冠有关键字 Private，则表示该成员是私有成员，它只能被本类声明的方法所访问，对其他类和方法是不可见的。

(2) Public (公有)。如果在成员变量前面冠有关键字 Private，则表示该成员是公有成员，它可以被程序中所有的方法访问，全透明。

(3) Protected (保护)。如果在成员变量前面冠有关键字 Protected，则表示该成员是保护成员，它只能在声明类和派生类中被访问。

(4) Published (发布型)。该关键字用于声明发布型成员，这类成员的可见性是最高的，不但在运行期间可见而且在设计期间也是可见的，这是 Delphi 默认的访问模式。

(5) Automated (自动型)。Automated 的作用与 Published 类似，不同的是，被声明为自动型的成员会生成自动类型信息，以用于创建 OLE 自动服务器等。

2. 类的继承性

类的构造可以在已有类的基础上进行，而不必对每一个类的构造都要从头开始。利用已有类来构造一个新类的过程，称为派生。如果类 A 是由类 B 派生而来的，则说类 A 是类 B 的继承类或者子类（也称扩展类、派生类），类 B 是类 A 的父类（也称超类），类 A 也说成是继承了类 B，继承关系可以用图 3.22 来表示。

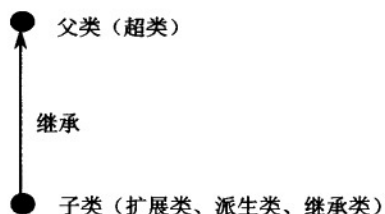


图 3.22 继承关系

在 Object Pascal 中，定义继承类（子类）的语法格式为：

```
type
```

```
    子类名 = class(父类名)
```

```
..... //成员的定义
```

```
end;
```

如果关键字 `class` 后面没有“(父类名)”，那么被定义的类也将自动地继承 Delphi 中最基本类——`TObject`。

子类将自动继承父类中的公有成员和保护成员。例如，要构建一个名为 `TEmployee` 类（即雇员类），使之继承 `TPerson` 类，则该类定义代码如下：

```
TEmployee=class(TPerson)
    gender:string[2];      //性别
    procedure SetPerson(Name,ID_Card,Address:String; Salary:currency;gender:String);
end;
```

该类中只定义了两个成员，但由于它继承了类 `TPerson`，所以还包含了 `Name`、`ID_Card`、`Address`、`Salary` 等成员。以下是类 `TEmployee` 中方法成员 `SetPerson` 的实现代码：

```
procedure TEmployee.SetPerson(Name,ID_Card,Address:String; Salary:currency;gender:String);
begin
    self.Name := Name;
    self.ID_Card := ID_Card;
    self.Address := Address;
    self.Salary := Salary;
    self.gender := gender;
end;
```

类 `TEmployee` 定义好后，就可以利用它。下面的程序就是一个运用类 `TEmployee` 的例子，希望读者能从中体会到类继承的基本用法。

```
program Project1;
{$APPTYPE CONSOLE}
uses
    SysUtils;
type
    TPerson=class
        Name:string;
        ID_Card:string;
        Address:string;
        Salary: currency;

        procedure SetPerson(Name,ID_Card,Address:String; Salary:currency);
    end;

    TEmployee=class(TPerson) //定义子类
        gender:string[2];
        procedure SetPerson(Name,ID_Card,Address:String; Salary:currency;gender:String);
    end;

var
    str:string;
    employee:TEmployee; //声明对象 employee
```

```
procedure TPerson.SetPerson(Name,ID_Card,Address:String; Salary:currency);
```

begin

```
self.Name := Name;
```

```
self.ID Card := ID Card;
```

```
self.Address := Address;
```

```
self.Salary := Salary;
```

end;

```
procedure TEmployee.SetPerson(Name,ID_Card,Address:String; Salary:currency;gender:String);
```

begin

```
self.Name := Name;
```

```
self.ID Card := ID Card;
```

```
self.Address := Address;
```

```
self.Salary := Salary;
```

```
self.gender := gender;
```

end:

begin

```
employee:=TEmployee.Create(); //创建对象
```

employee.SetPerson('黄蓉','1234567890','中南大学',2000.88,'女');

```
writeln('姓名: '+employee.name);
```

```
writeln('性别: '+employee.gender);
```

```
//writeln('姓名: '+employee.GetName());
```

```
writeln('身份证号码: '+employee.ID Card);
```

```
writeln('地址: '+employee.Address);
```

```
write('薪水: '); writeln(employee.Salary);
```

employee.Destroy;

```
readln(str); //让程序“停”下来,以观看结果
```

end.

该程序的运行结果如图 3.23 所示。

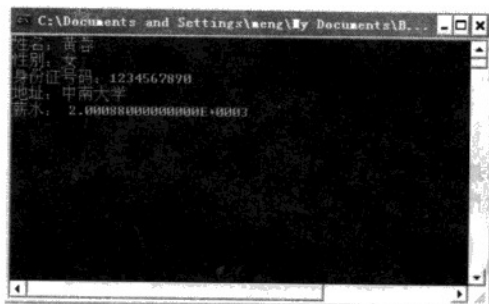


图 3.23 类继承实例程序的运行结果

容易看出，继承可以有效地提高代码的可重用性和程序的扩展性，减少代码开发成本。

3. 类的多态性

在 Object Pascal 中,类中的方法可以定义为下面三种方式之一:①静态方法;②虚拟方法;③动态方法。

前面介绍的方法的定义方式都是“静态”的，它们的调用地址在编译和连接的过程中就确定了。如果在一个类中定义了一个静态方法，而在它的子类中又定义了一个与该方法同名的方法，这就叫做静态重载，如果是虚拟方法或动态方法则叫做动态重载。

在上节的 TEmployee 类中的 SetPerson 方法的定义就是静态重载。

动态重载需要用到关键字 virtual 和 dynamic，前者用于指定一个方法为虚拟方法，后者则指定为动态方法。下面是一个虚拟方法重载的例子。

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  A = class    //父类
    procedure f(); virtual; //声明为虚拟方法
  end;

  B = class(A) //继承类（子类）
    procedure f(); override; //方法重载
  end;

procedure A.f();
begin
  writeln('A:在调用基类方法');
end;
procedure B.f();
begin
  writeln('B:在调用子类方法');
end;

var
  str:string;
  oa:A;    //声明对象 A
  ob:B;    //声明对象 B

begin
  oa:=A.Create; //创建对象 A
  ob:=B.Create; //创建对象 B
  oa.f();       //调用方法
  writeln("");  //隔行打印
  ob.f();       //调用方法
  oa.Destroy;   撤销对象
  ob.Destroy;
  readln(str);  //暂停
end.
```

一般来说，在静态方法不方便或者无法实现的情况下，可考虑运用动态方法重载。

3.8 小结

本章介绍了 Object Pascal 的程序设计方法，主要讲述了关于 Object Pascal 的发展、Object Pascal 数据类型、Object Pascal 的基本运算和流程控制语句，以及过程和函数的定义与调用，最后重点介绍了 Object Pascal 中 OOP 的编程技术，包括类与对象的基本概念、构造函数与析构函数、类的封装、继承和多态性等。通过本章的学习，读者应掌握：

- Object Pascal 语言的发展背景。
- Object Pascal 语言的基本语法，包括其数据类型、基本运算、流程控制语句的熟练运用。
- Object Pascal 面向对象的编程技术，包括对象与类的概念、类的声明和对象的创建和撤销方法，以及类的封装性、继承性和多态性。

第 4 章 Delphi 中消息和事件的处理

作为 Delphi 程序员,除了精通 Object Pascal 语言外,还要对 Delphi 的工作平台——Windows 操作系统及其与应用程序之间的运行机制有所了解。在 Delphi 中,大多数情况下 Windows 消息都被封装在 VCL (Visual Component Library) 的事件中,只需采用 Delphi 提供的可视化操作方式创建事件处理程序框架,然后往其中填写事件处理代码即可。但是,如果要深入剖析 Delphi 程序、写出高质量的 Delphi 应用程序,就必须对它的运行机制,包括其消息处理机制有较为深刻的理解。本章将重点研究 Delphi 程序对消息处理的一般过程、方法和机制等问题,同时包含许多实例,涉及的要点包括:

- Windows 消息的结构特征及其驱动机制。
- 在 Delphi 应用程序中捕获和处理 Windows 消息的途径和方法。
- Delphi 消息系统中的几种消息及其传递方法。
- 消息处理机制。

4.1 Windows 消息

Windows 系统是一个庞大的操作系统,其运行和工作机制非常复杂。它有一个特点非常明显,即“敌动我动,敌静我静”。也就是说,当单击鼠标、改变窗口尺寸或按下键盘上的一个键时,系统将作相应的状态改变,如运行程序、关闭窗口等;而如果没有对系统作任何“动作”时,它将“静静”地处于等待状态。实际上,Windows 是一个消息驱动 (Message Driven) 的操作系统。所谓消息,可以理解为 Windows 操作系统对应用程序发送的一个通知,告诉应用程序某个事情发生了。例如,单击鼠标、改变窗口尺寸、按下键盘上的一个键都会使 Windows 发送一个消息给应用程序 (如 Delphi 应用程序等),应用程序则对接收到的消息进行相应的处理,以完成既定的任务。

4.1.1 消息记录

在 Windows 操作系统中,消息是以一个记录的格式进行保存和传递的。消息记录包含了消息类型、消息参数、接受消息的应用程序窗口句柄及消息创建时间等。Windows 是采用类型为 TMsg 的记录来给 Delphi 应用程序发送消息的,该类型在 Delphi 的 Windows 单元中定义,格式如下:

```
tagMSG = packed record
    hwnd: HWND;
    message: UINT;
    wParam: WPARAM;
    lParam: LPARAM;
    time: DWORD;
    pt: TPoint;
```

```
end;
```

```
TMsg = tagMSG;
```

可见, 消息记录包含了六个字段, 现对其说明如下:

(1) hwnd. hwnd 代表接收消息的 32 位窗口句柄。窗口可以是任何类型的屏幕对象, 如应用程序窗口等。Win32 能够维护大多数可视对象的句柄, 如窗口、对话框、按钮、编辑框等。

(2) Message. message 为消息常量标识符, 用于区别其他消息的常量值, 这些常量可以是 Windows 单元中预定义的常量, 也可以是自定义的常量。

(3) wParam. wParam 通常是一个与消息有关的常量值 (32 位), 用于保存消息的附加信息, 也可能是窗口或控件的句柄。

(4) lParam. lParam 用于保存消息附加信息的 32 位消息参数, 通常是一个指向内存中数据的指针。

(5) time. time 记录了消息创建时的时间。

(6) pt. pt 用于保存消息创建时的鼠标位置。

Windows 消息主要分为两类: 标准 Windows 消息和用户定义的消息。不同的消息则由 message 的值 (消息常量) 来进行区分, 即不同的 message 值将确定不同的 Windows 消息。

标准 Windows 消息是指由 Windows 系统定义的、已经预先分配了消息常量值的消息。这些消息可以是硬件产生的输入消息, 也可以是系统窗口管理消息, 如关闭窗口消息、打开窗口消息等。这些消息都在 Delphi 的 Message 单元定义, 其含义如表 4.1 所示。

表 4.1 部分标准 Windows 消息及其含义

标准 Windows 消息	值	含义
WM_CREATE	\$0001	应用程序创建一个窗口
WM_DESTROY	\$0002	一个窗口被销毁
WM_MOVE	\$0003	移动一个窗口
WM_SIZE	\$0005	改变一个窗口的大小
WM_ACTIVATE	\$0006	一个窗口被激活或失去激活状态
WM_SETFOCUS	\$0007	获得焦点后
WM_KILLFOCUS	\$0008	失去焦点
WM_ENABLE	\$000A	改变 enable 状态
WM_SETREDRAW	\$000B	设置窗口是否能重画
WM_SETTEXT	\$000C	应用程序发送此消息来设置一个窗口的文本
WM_GETTEXT	\$000D	应用程序发送此消息来复制对应窗口的文本到缓冲区
WM_GETTEXTLENGTH	\$000E	得到与一个窗口有关的文本的长度 (不包含空字符)
WM_PAINT	\$000F	要求一个窗口重画自己
WM_CLOSE	\$0010	当一个窗口或应用程序要关闭时发送一个信号
WM_QUERYENDSESSION	\$0011	当用户选择结束对话框或程序自己调用 ExitWindows 函数
WM_QUIT	\$0012	用来结束程序运行或当程序调用 postquitmessage 函数
WM_QUERYOPEN	\$0013	当用户窗口恢复以前的大小位置时, 把此消息发送给某个图标

续表

标准 Windows 消息	值	含义
WM_ERASEBKGD	\$0014	当窗口背景必须被擦除时（如在窗口改变大小时）
WM_SYSCOLORCHANGE	\$0015	当系统颜色改变时，发送此消息给所有顶级窗口
WM_ENDSESSION	\$0016	当系统进程发出 WM_QUERYENDSESSION 消息后，此消息发送给应用程序，通知它对话是否结束
WM_SYSTEMERROR	\$0017	系统发生错误时产生此消息
WM_SHOWWINDOW	\$0018	当隐藏或显示窗口时发送此消息给这个窗口
WM_ACTIVATEAPP	\$001C	发此消息给应用程序哪个窗口是激活的，哪个是非激活的
WM_FONTCHANGE	\$001D	当系统的字体资源库变化时发送此消息给所有顶级窗口
WM_TIMECHANGE	\$001E	当系统的时间变化时发送此消息给所有顶级窗口
WM_CANCELMODE	\$001F	发送此消息来取消某种正在进行的模态（操作）
WM_SETCURSOR	\$0020	如果鼠标引起光标在某个窗口中移动且鼠标输入没有被捕获时，就发消息给某个窗口
WM_MOUSEACTIVATE	\$0021	当光标在某个非激活的窗口中而用户正按着鼠标的某个键发送此消息给当前窗口
WM_CHILDACTIVATE	\$0022	发送此消息给 MDI 子窗口当用户单击此窗口的标题栏，或当窗口被激活，移动，改变大小
WM_QUEUESYNC	\$0023	此消息由基于计算机的训练程序发送，通过 WH_JOURNAL-PALYBACK 的 hook 程序分离出用户输入消息
WM_GETMINMAXINFO	\$0024	此消息发送给窗口，当它将要改变大小或位置
WM_PAINTICON	\$0026	发送给最小化窗口，当它图标将要被重画
WM_ICONERASEBKGD	\$0027	此消息发送给某个最小化窗口，仅当它在画图标前它的背景必须被重画
WM_NEXTDLGCTL	\$0028	发送此消息给一个对话框程序去更改焦点位置
WM_SPOOLERSTATUS	\$002A	每当打印管理列队增加或减少一条作业时发出此消息
WM_DRAWITEM	\$002B	当 button、combobox、listbox 及 menu 的可视外观改变时发送此消息给这些控件的所有者
WM_MEASUREITEM	\$002C	当 button、combobox、listbox、listviewcontrol 及 ormenuitem 被创建时发送此消息给控件的所有者
WM_DELETEITEM	\$002D	当 listbox 或 combobox 被销毁或某些项被删除时产生该消息
WM_VKEYTOITEM	\$002E	此消息有一个 LBS_WANTKEYBOARDINPUT 风格的发出给它的所有者来响应 WM_KEYDOWN 消息
WM_CHARTOITEM	\$002F	此消息由一个 LBS_WANTKEYBOARDINPUT 风格的列表框发送给他的所有者来响应 WM_CHAR 消息
WM_SETFONT	\$0030	当绘制文本时程序发送此消息得到控件要用的颜色
WM_GETFONT	\$0031	应用程序发送此消息得到当前控件绘制文本的字体
WM_SETHOTKEY	\$0032	应用程序发送此消息让一个窗口与一个热键相关联
WM_GETHOTKEY	\$0033	应用程序发送此消息来判断热键与某个窗口是否有关联

续表

标准 Windows 消息	值	含义
WM_QUERYDRAGICON	\$0037	此消息发送给最小化窗口, 当此窗口将要被拖放而它的类中没有定义图标, 应用程序能返回一个图标或光标的句柄, 当用户拖放图标时系统显示这个图标或光标
WM_COMPAREITEM	\$0039	发送此消息来判定 combobox 或 listbox 新增项的相对位置
WM_COMPACTING	\$0041	显示内存已经很少了
WM_WINDOWPOSCHANGING	\$0046	发送此消息给那个窗口的大小和位置将要被改变时, 来调用 setwindowpos 函数或其他窗口管理函数
WM_WINDOWPOSCHANGED	\$0047	发送此消息给那个窗口的大小和位置已经被改变时, 来调用 setwindowpos 函数或其他窗口管理函数
WM_POWER	\$0048	当系统将要进入暂停状态时发送此消息 (适用于 16 位的 Windows)
WM_COPYDATA	\$004A	当一个应用程序传递数据给另一个应用程序时发送此消息
WM_CANCELJOURNAL	\$004B	当某个用户取消程序日志激活状态, 提交此消息给程序
WM_NOTIFY	\$004E	当某个控件的某个事件已经发生或这个控件需要得到一些信息时, 发送此消息给它的父窗口
WM_INPUTLANGCHANGE NGEREQUEST	\$0050	当用户选择某种输入语言或输入语言的热键改变时, 应用程序发送此消息
WM_INPUTLANGCHANGE	\$0051	当平台现场已经被改变后发送此消息给受影响的最顶级窗口
WM_TCARD	\$0052	当程序已初始化 Windows 帮助例程时, 发送此消息给应用程序
WM_HELP	\$0053	此消息显示用户按下了 F1, 如果某个菜单是激活的, 就发送此消息给此窗口关联的菜单, 否则就发送给有焦点的窗口, 如果当前都没有焦点, 就把此消息发送给当前激活的窗口
WM_USERCHANGED	\$0054	当用户已经登入或退出后发送此消息给所有的窗口, 当用户登入或退出时系统更新用户的具体设置信息, 在用户更新设置时系统马上发送此消息
WM_NOTIFYFORMAT	\$0055	公用控件, 自定义控件和及其父窗口通过此消息来判断控件是使用 ANSI 还是 UNICODE 结构发送 WM_NOTIFY 消息, 使用此控件能使某个控件与它的父控件之间进行相互通信
WM_CONTEXTMENU	\$007B	当用户某个窗口中单击了一下右键就发送此消息给这个窗口
WM_STYLECHANGING	\$007C	当调用 SETWINDOWLONG 函数将要改变一个或多个窗口的风格时发送此消息给那个窗口
WM_STYLECHANGED	\$007D	当调用 SETWINDOWLONG 函数一个或多个窗口的风格后发送此消息给那个窗口
WM_DISPLAYCHANGE	\$007E	当显示器的分辨率改变后发送此消息给所有的窗口
WM_GETICON	\$007F	此消息发送给某个窗口来返回与某个窗口有关联的大图标或小图标的句柄
WM_SETICON	\$0080	程序发送此消息让一个新的大图标或小图标与某个窗口关联
WM_NCCREATE	\$0081	当某个窗口第一次被创建时, 此消息在 WM_CREATE 消息发送前发送

续表

标准 Windows 消息	值	含义
WM_NCDESTROY	\$0082	此消息通知某个窗口, 非客户区正在销毁
WM_NCCALCSIZE	\$0083	当某个窗口的客户区域必须被核算时发送此消息
WM_NCHITTEST	\$0084	移动鼠标、按住或释放鼠标时产生此消息
WM_NCPAINT	\$0085	当窗口的框架必须被绘制时, 程序发送此消息给某个窗口
WM_NCACTIVATE	\$0086	此消息发送给某个窗口仅当它的非客户区需要被改变来显示是激活还是非激活状态
WM_GETDLGCODE	\$0087	发送此消息给某个与对话框程序关联的控件, Windows 控制方位键和 TAB 键使输入进入此控件通过响应 WM_GETDLGCODE 消息, 应用程序可以把它当成一个特殊的输入控件并能处理它
WM_NCMOUSEMOVE	\$00A0	当光标在一个窗口的非客户区内移动时发送此消息给这个窗口, file://非客户区为: 窗体的标题栏及窗的边框体
WM_NCLBUTTONDOWN	\$00A1	当光标在一个窗口的非客户区同时按鼠标左键时提交此消息
WM_NCLBUTTONUP	\$00A2	当用户释放鼠标左键同时光标某个窗口在非客户区时发送此消息
WM_NCLBUTTONDBLCLK	\$00A3	当用户双击鼠标左键同时光标某个窗口在非客户区时发送此消息
WM_NCRBUTTONDOWN	\$00A4	当用户按下鼠标右键同时光标又在窗口的非客户区时发送此消息
WM_NCRBUTTONUP	\$00A5	当用户释放鼠标右键同时光标又在窗口的非客户区时发送此消息
WM_NCRBUTTONDBLCLK	\$00A6	当用户双击鼠标右键同时光标某个窗口在非客户区时发送此消息
WM_NCMBUTTONDOWN	\$00A7	当用户按下鼠标中键同时光标又在窗口的非客户区时发送此消息
WM_NCMBUTTONUP=\$00A8;	\$00A8	当用户释放鼠标中键同时光标又在窗口的非客户区时发送此消息
WM_NCMBUTTONDBLCLK	\$00A9	当用户双击鼠标中键同时光标又在窗口的非客户区时发送此消息
WM_KEYDOWN	\$0100	按下一个键
WM_KEYUP	\$0101	释放一个键
WM_CHAR	\$0102	按下某键, 并已发出 WM_KEYDOWN、WM_KEYUP 消息
WM_DEADCHAR	\$0103	当用 TRANSLATEMESSAGE 函数翻译 WM_KEYUP 消息时发送此消息给拥有焦点的窗口
WM_SYSKEYDOWN	\$0104	当用户按住 ALT 键同时按下其他键时提交此消息给拥有焦点的窗口
WM_SYSKEYUP	\$0105	当用户释放一个键同时 ALT 键还按着时提交此消息给拥有焦点的窗口
WM_SYSCHAR	\$0106	当 WM_SYSKEYDOWN 消息被 TRANSLATEMESSAGE 函数翻译后提交此消息给拥有焦点的窗口
WM_SYSDEADCHAR	\$0107	当 WM_SYSKEYDOWN 消息被 TRANSLATEMESSAGE 函数翻译后发送此消息给拥有焦点的窗口
WM_INITDIALOG	\$0110	在一个对话框程序被显示前发送此消息给它, 通常用此消息初始化控件和执行其他任务
WM_COMMAND	\$0111	当用户选择一条菜单命令项或当某个控件发送一条消息给它的父窗口, 一个快捷键被翻译

续表

标准 Windows 消息	值	含义
WM_SYSCOMMAND	\$0112	当用户选择窗口菜单的一条命令或当用户选择最大化或最小化时那个窗口会收到此消息
WM_TIMER	\$0113	发生了定时器事件
WM_HSCROLL	\$0114	当一个窗口标准水平滚动条产生一个滚动事件时发送此消息给那个窗口, 也发送给拥有它的控件
WM_VSCROLL	\$0115	当一个窗口标准垂直滚动条产生一个滚动事件时发送此消息给那个窗口, 也发送给拥有它的控件
WM_INITMENU	\$0116	当一个菜单将要被激活时发送此消息, 它发生在用户菜单条中的某项或按下某个菜单键, 它允许程序在显示前更改菜单
WM_INITMENUPOPUP	\$0117	当一个下拉菜单或子菜单将要被激活时发送此消息, 它允许程序在它显示前更改菜单, 而不要改变全部
WM_MENUSELECT	\$011F	当用户选择一条菜单项时发送此消息给菜单的所有者(一般是窗口)
WM_MENUCHAR	\$0120	当菜单已被激活用户按下了某个键(不同于加速键), 发送此消息给菜单的所有者
WM_ENTERIDLE	\$0121	当一个模态对话框或菜单进入空载状态时发送此消息给它的所有者, 一个模态对话框或菜单进入空载状态就是在处理完一条或几条先前的消息后若没有消息则它在列队中等待

除了标准 Windows 消息外, 用户也可以定义自己的消息常量, 相应的消息就称为用户定义的消息, 以及通知消息等。但是, Windows 系统为了对消息进行有效管理, 避免使用上的冲突, 对各类消息常量的取值范围进行了限定, 限定情况如表 4.2 所示。

表 4.2 消息常量取值的限定

取值范围	含义
\$0000~\$03FF	标准 Windows 消息常量, 用户不能使用
\$0400~\$7FFF	可以在私有窗口类中定义使用的消息常量
\$8000~\$BFFF	Windows 保留使用, 用于今后使用的消息常量
\$C000~\$FFFF	应用程序使用的字符串消息常量
\$FFFF	Windows 保留使用

4.1.2 消息的驱动机制

Windows 消息提供了应用程序与应用程序以及应用程序与 Windows 系统之间进行通信的手段。Windows 系统本身维护一个存放消息的消息队列, 称为系统消息队列。另外, 对于每一个正在执行的 Windows 应用程序, Windows 系统为其建立一个“消息队列”, 即应用程序消息队列, 用来存放该程序可能创建的各种窗口的消息。当某一事件发生时, Windows 将会生成相应的消息, 并更新系统消息队列, 同时把这些消息放入相应的应用程序消息队列。

每一个应用程序中都包含一段用于从消息队列中检索这些消息并把它们分发到相应的窗口函数中去的代码, 这段代码称为消息循环代码。消息循环代码的功能具有循环性, 即不断地

检索应用程序消息队列中的消息，并进行分发，直到收到退出程序的消息为止。在程序代码形式上，消息循环代码是应用程序中主函数 WinMain () 中类似如下的程序段：

```
while(GetMessage(&&msg,NULL,NULL,NULL)) //从应用程序消息队列中取得消息
{
    TranslateMessage(&&msg); //检索并生成字符消息 WM_CHAR
    DispatchMessage(&&msg); //将消息发送给相应的窗口函数
}
```

Windows 应用程序在创建每个窗口时都会生成并在系统核心注册一个相应的窗口函数。窗口函数用于处理由消息循环发送到该窗口的消息，它由 Windows 采用消息驱动的形式直接调用，而不是由应用程序显示调用的。窗口函数处理完消息后又将控制权返回给 Windows。

系统消息队列、应用程序消息队列、消息循环和窗口函数之间的关系如图 4.1 所示。

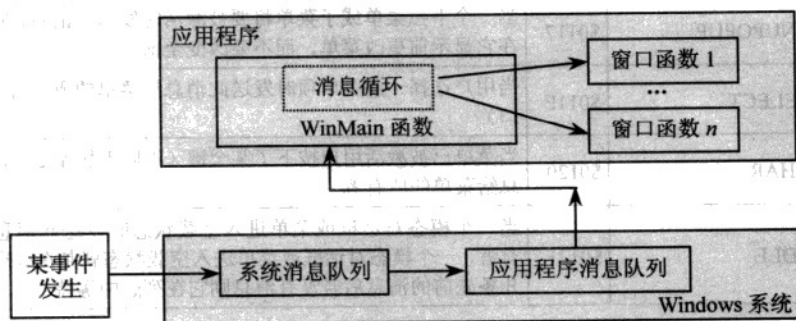


图 4.1 系统消息队列、应用程序消息队列、消息循环和窗口函数之间的关系

4.2 Delphi 中捕获和处理 Windows 消息

Delphi 应用程序作为基于 Windows 平台上的应用程序，其运行当然离不开 Windows 消息的应用，但是在编写 Delphi 程序时几乎都没有涉及 Windows 消息编程。其主要原因是，Delphi 的 VCL (Visual Component Library) 事件系统对许多 Windows 消息进行完美的包装 (封装)，实际上 Delphi 的 VCL 事件系统正是为了更好地与 Windows 消息接口而设计的。这样，使程序员不但可以方便地截获 Windows 消息，并根据需要发送和处理消息，而且避免了编写烦琐的处理代码的过程，提高了程序的可读性，减少了代码出错率。

Delphi 对 Windows 消息进行了细致的封装，Delphi 程序员根本就不需直接接触 Windows 消息。循环代码则封装在 TApplication 类中，该类提供了方便简捷的消息传递和处理方法。以下代码是 Delphi 应用程序工程文件的核心代码：

```
begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

其中，TApplication 类的 Initialize 方法用于完成初始化工作，CreateForm 则用于创建窗体类，Run 方法用于运行程序。

使用消息就是在对 Windows 消息进行响应和处理, 这需要编写代码来完成。由于 Delphi 提供了 VCL 事件系统, 对窗口过程进行了封装, 所以在 Delphi 中使用 Windows 消息已变得非常方便。下面介绍在 Delphi 中几种使用 Windows 的方法。

1. 自定义消息处理方法

在 Delphi 中, 也完全可以通过自己定义相应的消息处理过程或方法来完成对 Windows 消息的响应和处理。但在此之前, 需要了解 Delphi 对 Windows 消息的封装原理。

对于对 Windows 消息的封装, 可以分为通用消息记录的封装和专用消息记录的封装。

下面是通用的 Windows 消息记录的声明:

```
tagMSG = packed record
  hwnd: HWND;
  message: UINT;
  wParam: WPARAM;
  lParam: LPARAM;
  time: DWORD;
  pt: TPoint;
end;
{$EXTERNALSYM tagMSG}
TMsg = tagMSG;
```

在 Delphi 中则把它映射为如下的消息记录 TMessage:

```
TMessage = packed record
  Msg: Cardinal;
  case Integer of
    0: (
      wParam: Longint;
      lParam: Longint;
      Result: Longint);
    1: (
      WParamLo: Word;
      WParamHi: Word;
      LParamLo: Word;
      LParamHi: Word;
      ResultLo: Word;
      ResultHi: Word);
  end;
```

其中, TMessage 中字段 Msg 对应于 Windows 消息 TMsg 的消息常量 message, 用于区分不同类型的消息; WParamLo 和 WParamHi 则分别对应 Windows 消息中 wParam 字段的低位和高位等。另外, 在 TMessage 记录中还包含了 Result 字段, 它用于保存返回值。这样, 在需要返回值的时候, 只要对 Result 字段赋值即可。类似地, ResultLo 和 ResultHi 分别表示 Result 字段的低位和高位, 访问时需分别对待。

回顾前面的内容可知, 除了通用的 Windows 消息记录 TMsg 外, Windows 消息还包含许多专用的消息, 如图点击鼠标所产生的 Windows 消息, 按键盘时产生的 Windows 消息等。对于这些专用消息, Delphi 也几乎都作了封装和映射, 这可以更加方便地引用消息中包含的信息。

例如, 在 Delphi 中专用的鼠标消息记录定义如下:

```

TWMMouse = packed record
  Msg: Cardinal;
  Keys: Longint;
  case Integer of
    0: (
      XPos: Smallint;
      YPos: Smallint);
    1: (
      Pos: TSmallPoint;
      Result: Longint);
  end;

```

并把标准 Windows 消息 WM_LButtonDblClk 等映射为 TWMMouse 消息:

```

TWMLButtonDblClk = TWMMouse;
TWMLButtonDown = TWMMouse;
TWMLButtonUp = TWMMouse;
TWMMButtonDblClk = TWMMouse;
TWMMButtonDown = TWMMouse;
TWMMButtonUp = TWMMouse;

```

利用这些映射关系, 可以定义相应的消息处理过程, 即通常所称的消息处理方法。消息处理就是定义应用程序如何响应 Windows 的消息。在 Delphi 中每一个消息都有自己的处理过程, 它必须是一个对象中的方法, 且只能传递一个 TMessage 或其他特殊的消息记录, 方法声明后要有一个 message 命令, 后接一个在 0 到 32767 之间的常量。具体地讲, 对于消息处理方法定义需要注意以下几点: ①过程定义时必须包含一个变量参数 (由 var 说明), 该参数是 TMessage 或者专用消息记录类型; ②在过程声明语句后面, 必须使用 message 关键字, 其后跟所处理消息的 Windows 消息常量; ③该过程必须为某一个对象的方法, 通常为私有方法。下面是消息处理方法定义的一般格式:

```

procedure 方法名(var 消息变量: 消息类型); message Windows 消息常量;

```

下面的例子是通过自定义消息处理方法来捕获 Windows 消息并进行相应的处理。

例 4.1 双击鼠标左键和移动鼠标时会分别产生 WM_LButtonDblClk 消息和 WM_MOUSEMOVE 消息 (Windows 消息)。下面编写两个自定义消息处理方法, 它们可以分别捕获这两个消息, 并在列表框中打印出双击鼠标时鼠标所在位置的坐标信息, 以及在文本框中打印当前鼠标的实时位置坐标信息。

(1) 创建一个新的应用程序, 将工程命名为 Ex4_1.dpr, 主窗体单元文件采用默认的名称——Unit1.pas。然后在主窗体中添加一个 ListBox 组件和一个 Edit 组件, 分别采用默认名称 ListBox1 和 Edit1。

(2) 在单元文件 Unit1.pas 中, 对窗体类 TForm1 的 private 部分中声明两个自定义消息处理方法:

```

procedure WMLButtonDown(var Msg1: TWMMouse); message WM_LButtonDblClk;

```

```

procedure WMLMouseMove(var Msg2: TWMMouse); message WM_MOUSEMOVE;

```

它们分别用于捕获 WM_LButtonDblClk 消息和 WM_MOUSEMOVE 消息, 并进行相应的处理 (在此主要是打印鼠标的位置信息)。

(3) 在 implementation 部分, 对上述声明的两个方法予以实现, 代码如下:

```
procedure TForm1.WMLButtonDown(var Msg1: TWMMouse);
begin
    ListBox1.Items.Add('双击时鼠标位置: ('+IntToStr(Msg1.XPos)+'/'+IntToStr(Msg1.YPos)+'');
    inherited;
end;

procedure TForm1.WMLMouseMove(var Msg2: TWMMouse);
begin
    Edit1.Text:='当前鼠标位置: ('+IntToStr(Msg2.XPos)+'/'+IntToStr(Msg2.YPos)+'');
end;
```

运行该程序, 当鼠标在主窗体上移动时, Windows 系统会产生 WM_MOUSEMOVE 消息, 并把它分发给主窗体; 主窗体类中定义的方法 (过程) WMLMouseMove 会捕获该 Windows 消息并把它映射为 Delphi 的专用消息——TWMMouse 消息; 然后在处理 TWMMouse 消息的过程中, 打印出该消息的 XPos 和 YPos 字段, 即鼠标的坐标信息, 如图 4.2 所示。

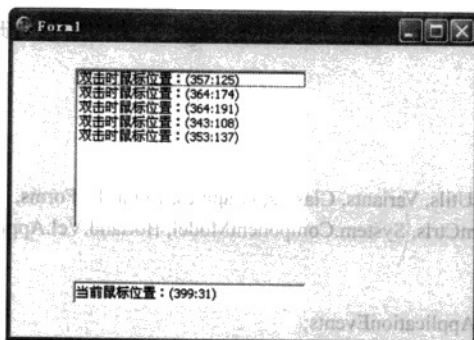


图 4.2 程序 Ex4_1 的运行结果

添加语句 “inherited;” 的目的是使主窗体的 OnDbClick 方法变得有效, 否则在执行 WMLButtonDown 方法后即终止, 而使得主窗体的 OnDbClick 方法变得无效。

2. 利用 Application 对象的 OnMessage 方法

TApplication 类封装了循环代码, 提供了许多消息传递和处理的方法。OnMessage 方法就是其中之一, 可以利用该方法来捕获和处理 Windows 消息。

在 Delphi 2005 中, 提供了 ApplicationEvents 组件, 该组件包含 OnMessage 方法, 通过该方法可以直接编写 Application 对象的消息处理过程。下面通过一个例子来说明这种方法。

例 4.2 当单击鼠标左键或按下键盘上某一个键时都会产生 Windows 消息。下面的程序将显示其收到的消息总数、消息常量值、参数 wParam 和 lParam 的值, 以及消息产生时鼠标所在的位置等信息。

(1) 创建一个新的 VCL Forms 应用程序, 工程名取 Ex4_2.dpr, 主窗体单元文件名取默认值 Unit1.pas。

(2) 在主窗体上添加 ApplicationEvents 组件和 ListView 组件, 生成的对象分别命名为 ApplicationEvents1 和 ListView1, 各组件属性的设置情况如表 4.3 所示。

表 4.3 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
ListView	ListView1	Align	alClient
		ViewStyle	vsReport
		Columns[0].Caption	消息计数
		Columns[1].Caption	消息常量值
		Columns[2].Caption	wParam 的值
		Columns[3].Caption	lParam 的值
		Columns[4].Caption	产生消息的时间
		Columns[5].Caption	鼠标的 X 坐标值
		Columns[6].Caption	鼠标的 Y 坐标值
ApplicationEvents	ApplicationEvents1		

(3) 编写 ApplicationEvents1 对象的 OnMessage 方法的事件处理代码, 结果得到的代码 (Unit1.pas 文件) 如下:

```

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Borland.Vcl.ComCtrls, System.ComponentModel, Borland.Vcl.AppEvnts;
type
  TForm1 = class(TForm)
    ApplicationEvents1: TApplicationEvents;
    ListView1: TListView;
    procedure ApplicationEvents1Message(var Msg: tagMSG; var Handled: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  MsgCount: Integer;

implementation

{$R *.nfm}

procedure TForm1.ApplicationEvents1Message(var Msg: tagMSG;
  var Handled: Boolean);

var ListItem: TListItem;
begin

```

```

Inc(MsgCount);
case Msg.message of
  WM_KEYDOWN:      //处理按键消息
  begin
    ListItem := ListView1.Items.Add;
    ListItem.Caption := IntToStr(MsgCount);           //消息计数
    ListItem.SubItems.Add(Format('%x',[Msg.message])); //常量消息
    ListItem.SubItems.Add(Format('%s',[Chr(Msg.wParam)])); //参数 wParam 值
    ListItem.SubItems.Add(Format('%x',[Msg.lParam]])); //参数 lParam 值
    ListItem.SubItems.Add(Format('%d',[Msg.time])); //产生消息的时间
    ListItem.SubItems.Add(Format('%d',[Msg.pt.X])); //鼠标的 X 坐标值
    ListItem.SubItems.Add(Format('%d',[Msg.pt.Y])); //鼠标的 Y 坐标值
  end;
  WM_LBUTTONDOWN: //处理按鼠标产生的消息
  begin
    ListItem := ListView1.Items.Add;
    ListItem.Caption := IntToStr(MsgCount);           //消息计数
    ListItem.SubItems.Add(Format('%x',[Msg.message])); //常量消息
    ListItem.SubItems.Add(Format('%x',[Msg.wParam]])); //参数 wParam 值
    ListItem.SubItems.Add(Format('%x',[Msg.lParam]])); //参数 lParam 值
    ListItem.SubItems.Add(Format('%d',[Msg.time])); //产生消息的时间
    ListItem.SubItems.Add(Format('%d',[Msg.pt.X])); //鼠标的 X 坐标值
    ListItem.SubItems.Add(Format('%d',[Msg.pt.Y])); //鼠标的 Y 坐标值
  end;
end;
Handled:=false;
end;
end.

```

(4) 运行该程序, 结果如图 4.3 所示。注意到, 消息计数之间的整数是不连续的。例如, 第一行是 115, 而第二行是 128。这说明在显示的两条消息之间还捕获到了其他的一些消息, 但由于程序中只有两个消息处理代码, 所以对于捕获到的其他消息并没有在程序运行界面中得到反映。

消息计数	消息常量值	wParam值	lParam值	产生消息的时间	鼠标位置的x坐标值	鼠标位置的y坐标值
115	201	1	80031	9669063	187	214
128	201	1	80031	9669714	187	214
244	201	1	70077	9672147	257	213
379	201	1	600c1	9675763	331	212
496	201	1	5010f	9678056	409	211
617	201	1	4015d	9680690	487	210
743	201	1	80180	9683203	562	214
882	201	1	9023e	9686488	712	215
1221	100	K	250001	9703573	649	428
1232	100	J	240001	9704203	649	428
1283	100	G	220001	9708760	649	428
1291	100	H	230001	9709100	649	428
1298	100	H	230001	9709251	649	428
1454	100	K	250001	9720256	716	454
1467	100	H	230001	9721118	716	454

图 4.3 程序 Ex4_2.dpr 的运行结果

4.3 Delphi 消息系统

Windows 消息是由外部事件或 Windows 系统本身的管理事件而引发。除此之外, Delphi 在 Controls 单元中也定义了许多消息, 通常称为 VCL 内部消息; 用户也可以根据自己定义相应的消息, 称为自定义消息。对于这些消息, 也可以进行捕获和处理。作为 Delphi 程序员, 了解和掌握这些消息的定义、捕获和处理方法, 对于更好地开发 VCL 组件也是很重要的。

4.3.1 VCL 内部消息

前面提到的消息都是标准 Windows 消息, 其常量标识符是以“WM_”开头的。对 VCL 内部消息, 其常量标识符通常以“CM_”开头, 用于管理 VCL 内部的事物。如果改变了组件的某个属性值或其他一些特性后, 需要通过内部消息将该变化通知其他组件。例如, 激活输入焦点消息是向被激活的或被停用的组件发送的, 用于接受或放弃输入焦点。

VCL 内部消息在 Controls 单元中定义, 例如, 下面都是 Delphi 定义的部分 VCL 内部消息:

```
CM_BASE           = $B000;
CM_ACTIVATE       = CM_BASE + 0;
CM_DEACTIVATE     = CM_BASE + 1;
CM_GOTFOCUS       = CM_BASE + 2;
CM_LOSTFOCUS      = CM_BASE + 3;
CM_CANCELMODE     = CM_BASE + 4;
CM_DIALOGKEY      = CM_BASE + 5;
CM_DIALOGCHAR     = CM_BASE + 6;
CM_FOCUSCHANGED   = CM_BASE + 7;
CM_PARENTFONTCHANGED = CM_BASE + 8;
CM_PARENTCOLORCHANGED = CM_BASE + 9;
CM_HITTEST        = CM_BASE + 10;
CM_VISIBLECHANGED = CM_BASE + 11;
CM_ENABLEDCHANGED = CM_BASE + 12;
CM_COLORCHANGED   = CM_BASE + 13;
CM_FONTCHANGED    = CM_BASE + 14;
CM_CURSORCHANGED  = CM_BASE + 15;
CM_CTL3DCHANGED   = CM_BASE + 16;
CM_PARENTCTL3DCHANGED = CM_BASE + 17;
CM_TEXTCHANGED    = CM_BASE + 18;
CM_MOUSEENTER     = CM_BASE + 19;
CM_MOUSELEAVE     = CM_BASE + 20;
```

下面通过一个例子来说明 VCL 内部消息的使用方法。

例 4.3 在 Delphi 中, 鼠标移进 VCL 组件上方时都会产生 CM_MOUSEENTER 消息, 而移出组件上方时则会产生 CM_MOUSELEAVE 消息。下面的程序中将编写一个消息处理方法, 使得当鼠标移进 ListBox 组件时在该组件中显示“鼠标移入列表框...”, 而当鼠标移出时则在该组件中显示“鼠标移出列表框...”。

(1) 创建一个新的 VCL Forms 应用程序, 工程名取为 Ex4_3.dpr, 主窗体单元文件名取

默认值 Unit1.pas。

(2) 在 interface 部分定义类 TMyListBox, 该类继承类 TListBox, 并在该类中声明两个方法: MouseIntoListbox 和 MouseFormListbox, 它们分别是鼠标移入和移出 ListBox 组件时产生消息的消息处理方法。另外, 还在类 TMyListBox 中声明一个方法——DrawInListBox, 它用于向 ListBox 组件加入提示信息。类 TMyListBox 的定义代码如下:

```
TMyListBox = class(TListBox)
private
    { Private declarations }
    procedure MouseIntoListbox(var Msg:TMessage); message CM_MOUSEENTER;
    procedure MouseFormListbox(var Msg:TMessage); message CM_MOUSELEAVE;
    procedure DrawInListBox(Color: TColor; const s:string);
public
    { Public declarations }
end;
```

(3) 在 implementation 部分实现 DrawInListBox 方法, 它用于向 ListBox 添加相应的提示信息, 并且不同的信息采用不同的颜色, 其实现代码如下:

```
procedure TMyListBox.DrawInListBox(Color: TColor; const s:string);
begin
    MyListBox.Canvas.Font.Color := Color; //设置颜色
    MyListBox.Canvas.Font.Size := 13;    //设置字体
    MyListBox.Canvas.Font.Style := [fsBold]; //加粗显示
    MyListBox.Canvas.TextOut(10,row,s); //打印信息
    Inc(row,25);
end;
```

(4) 在 implementation 部分实现消息处理方法——MouseIntoListbox 方法和 MouseFormListbox 方法, 实现代码如下:

```
procedure TMyListBox.MouseFormListbox(var Msg:TMessage);
begin
    DrawInListBox(clGreen,'鼠标移入列表框...');
end;

procedure TMyListBox.MouseIntoListbox(var Msg:TMessage);
begin
    DrawInListBox(clNavy,'鼠标移出列表框...');
end;
```

(5) 定义全局变量 MyListBox 和 row:

```
var
    Form1: TForm1;           //Delphi 自动定义
    MyListBox: TMyListBox;   //新定义
    row: Integer;            //新定义
```

(6) 在窗体的 OnCreate 事件处理程序中创建并显示 ListBox 对象实例, 其代码如下:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    MyListBox := TMyListBox.Create(Self);
```



```

MyListBox.Parent := Self;
MyListBox.Width := Self.ClientWidth div 2;
MyListBox.Align := alLeft;
row := 10;
end;

```

(7) 编译、运行该程序，当鼠标移入 ListBox 对象上方时将在 ListBox 中显示“鼠标移入列表框...”，移出时则显示“鼠标移出列表框...”，如图 4.4 所示。

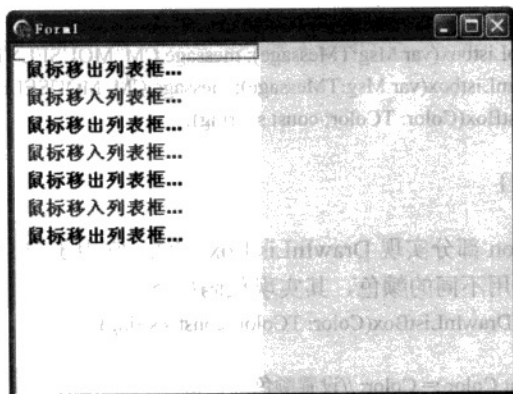


图 4.4 程序 Ex4_3.dpr 的运行结果

4.3.2 通知消息

有一种消息称为通知消息，它用于将一个窗口中组件发生的事情通知父窗口，但只适用于标准窗口中的组件，如按钮、列表框、编辑框等。通知消息在 Message 单元中声明：

```

BN_CLICKED           = 0;
BN_PAINT             = 1;
BN_HILITE            = 2;
BN_UNHILITE          = 3;
BN_DISABLE           = 4;
BN_DOUBLECLICKED     = 5;
BN_PUSHED            = BN_HILITE;
BN_UNPUSHED          = BN_UNHILITE;
BN_DBLCLK            = BN_DOUBLECLICKED;
BN_SETFOCUS          = 6;
BN_KILLFOCUS         = 7;

```

其中，前面三个分别是按钮的通知消息，表示用户单击了按钮、按钮应当重画、用户加亮了按钮。

4.3.3 自定义消息与消息的发送

在程序设计过程中，有时根据需要程序员也可以自己定义消息（称为自定义消息）。与其他消息一样，自定义消息也可以进行发送、编写消息处理方法等。但要注意的是，Windows 系统已经规定了自定义消息的常量值范围，即 WM_USER+100 到 \$7FFF (WM_USER = \$0400)，

如果超出了这个范围将与其他消息常量值相冲突。

理论上, 自己定义消息的常量标识符可以采用任何合法的 Delphi 标识符, 但习惯上是采用以“SX_”开头的大写 Delphi 标识符, 如

```
const
  SX_MESSAGE1 = WM_USER + 200;
... ..
```

Windows 消息是由 Windows 系统调用相应的方法自动进行发送的。显然, 被定义自定义消息也只有被发送才有意义。那么, 如何发送自定义消息呢? 一般来说, 自定义消息的发送有三种方法: TControl 类的 Perform 对象方法、Windows 的 API 函数 SendMessage() 和 Postmessage(), 以及 TWinControl 类的 Broadcast 方法(消息广播)。当然这几种方法对一般 Windows 消息也适用, 不过这时是对 Windows 消息进行转发或把它当作自定义消息来使用。

1. Perform 方法

Perform 方法是 Tcontrol 类的对象方法, 它可向任意窗体或控件发送消息。Perform 方法声明如下:

```
function Tcontrol.Perform(Msg: Cardinal; Wparam, Lparam: Longint): Longint
```

它是 Tcontrol 类的公有成员函数, 所以 Tcontrol 的任何派生类都继承了该方法, 如 TForm 类就是 Tcontrol 类的一个派生类, 因此可以调用对象方法 Form1.Perform() (Form1 为 TForm 类对象) 进行消息发送。

2. SendMessage() 和 PostMessage() 方法

这两个函数都是 Windows 的 API 函数, 分别声明如下:

```
function SendMessage(hWnd: HWND;
  Msg: UINT;
  wParam: WPARAM;
  lParam: LPARAM): LRESULT; stdcall;
```

```
function PostMessage(hWnd: HWND;
  Msg: UINT;
  wParam: WPARAM;
  lParam: LPARAM): BOOL; stdcall;
```

从上述定义可知, 这两个函数的参数是相同的, 其中 hWnd 为接受消息的窗口句柄, Msg 为消息常量, wParam 和 lParam 分别为附加的两个参数。

这两个 API 函数及 Perform 函数之间的差别在于, PostMessage 函数先把消息添加到应用程序的消息队列中去, 然后经过消息循环再从消息队列中提取消息并进行登记, 最后发送到相应的窗口中; 而 SendMessage 函数则越过消息队列直接向窗口过程发送消息, 不必经过循环等待。在这一点上, Perform 方法与 SendMessage 函数类似, 但是 SendMessage 函数(及 PostMessage 函数)可以向任何一个窗口(不限于 Delphi 应用程序窗口, 只要知道窗口句柄就行)发送, 而 Perform 要发送的前提是已经知道窗体或控件的实例, 否则不能发送。

3. Broadcast 方法

该方法是 TWinControl 类的一个方法, 它以广播的方式发送消息。其声明如下:

```
procedure Broadcast(var Message);
```

该方法的参数简单, 发送时不需要知道窗口句柄和对象实例, 但通常限于窗口中的组件,

而窗口本身则不能运用该方法发送消息。

下面将通过例子说明如何结合消息的发送方法来运用自定义消息。

例 4.4 利用 Windows 消息和用户自定义消息，实现在一个文本编辑框中输入一个字母，使得它同时在其他若干个文本编辑框中被显示。

回顾前面的内容可知，按下键盘上的任一键都会产生 WM_KEYDOWN 消息。可以捕获该消息，并在消息处理方法中用 Broadcast 方法对其进行广播，使得该消息被其他若干个文本编辑框捕获并显示。下面的程序正是基于这样的思路而设计的，步骤如下。

(1) 创建一个新的 VCL Forms 应用程序，工程命名为 Ex4_4.dpr，主窗体单元文件名取默认值 Unit1.pas。

(2) 在窗体上添加 Panel 组件，name 属性值设为 Panel1，Align 属性值为 alClient，Caption 属性值为空值。

(3) 在 interface 部分先声明自定义常量 SX_MESSAGE1：

```
const
```

```
  SX_MESSAGE1 = WM_USER + 200;
```

然后定义类 TMyEdit 和类 TInputEdit，它们都继承类 TEdit，并在其中分别声明消息处理方法 SXMessage1 和 WMKeyDown：

```
TMyEdit = class(TEdit)
```

```
private
```

```
  { Private declarations }
```

```
  procedure SXMessage1(var Msg:TMessage); message SX_MESSAGE1;
```

```
public
```

```
  { Public declarations }
```

```
end;
```

```
TInputEdit = class(TEdit)
```

```
private
```

```
  { Private declarations }
```

```
  procedure WMKeyDown(var Msg:TMessage); message WM_KEYDOWN;
```

```
public
```

```
  { Public declarations }
```

```
end;
```

其中，WMKeyDown 用于捕获键盘消息 WM_KEYDOWN（属于 Windows 消息）并转化为自定义消息进行转发，SXMessage1 方法则用于捕获自定义消息 SX_MESSAGE1 并显示被按下的键盘字母。这两个方法的实现代码如下：

```
procedure TInputEdit.WMKeyDown(var Msg:TMessage);
```

```
var MsgRec : TMessage;
```

```
begin
```

```
  MsgRec := TMessage.Create();
```

```
  with MsgRec do
```

```
  begin
```

```
    Msg := SX_MESSAGE1;
```

```

end;
MsgRec.WParam := Msg.WParam;
Form1.Panel1.Broadcast(MsgRec);
end;

procedure TMyEdit.SXMessage1(var Msg:TMessage);
begin
    self.Text := self.Text + chr(Msg.WParam);
end;

```

(4) 在 interface 部分声明类 TMyEdits 和类 TInputEdit 的变量, 代码如下:

```

var
    Form1: TForm1;
    MyEdits : array[0..4] of TMyEdit;
    InputEdit : TInputEdit;

```

(5) 在窗体的 OnCreate 事件处理程序中创建类 TMyEdit 和类 TInputEdit 的对象实例, 代码如下:

```

procedure TForm1.FormCreate(Sender: TObject);
var i:integer;
begin
    for i:= 0 to 4 do
    begin
        MyEdits[i] := TMyEdit.Create(self);
        MyEdits[i].Parent := Panel1;
        MyEdits[i].Name := 'MyEdit'+IntToStr(i);
        MyEdits[i].Left := 4;
        MyEdits[i].Width := self.ClientWidth - 4;
        MyEdits[i].Height := self.ClientHeight div 8;
        MyEdits[i].Top := i*MyEdits[i].Height;
        MyEdits[i].Text := "";
    end;
    InputEdit := TInputEdit.Create(self);
    InputEdit.Parent := Panel1;
    InputEdit.Left := 4;
    InputEdit.Color := RGB(0,200,100);
    InputEdit.Top := 6*MyEdits[4].Height;
end;

```

(6) 经过上述的编码过程后, 就形成了一个完整的程序, 其代码如下:

```

unit Unit1;
interface

uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, System.ComponentModel, Borland.Vcl.StdCtrls, Borland.Vcl.ExtCtrls;
const
    SX_MESSAGE1 = WM_USER + 200;

```

```

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  TMyEdit = class(TEdit)
  private
    { Private declarations }
    procedure SXMessage1(var Msg:TMessage); message SX_MESSAGE1;
  public
    { Public declarations }
  end;
  TInputEdit = class(TEdit)
  private
    { Private declarations }
    procedure WMKeyDown(var Msg:TMessage); message WM_KEYDOWN;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  MyEdits : array[0..4] of TMyEdit;
  InputEdit : TInputEdit;
implementation
{$R *.nfm}

procedure TInputEdit.WMKeyDown(var Msg:TMessage);
var MsgRec : TMessage;
begin
  MsgRec := TMessage.Create();
  with MsgRec do
  begin
    Msg := SX_MESSAGE1;
  end;
  MsgRec.WParam := Msg.WParam;
  Form1.Panel1.Broadcast(MsgRec);
end;

procedure TMyEdit.SXMessage1(var Msg:TMessage);
begin
  self.Text := self.Text + chr(Msg.WParam); //IntToStr(Msg.WParam);

```

```
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
```

```
var i:integer;
```

```
begin
```

```
  for i:= 0 to 4 do
```

```
  begin
```

```
    MyEdits[i] := TMyEdit.Create(self);
```

```
    MyEdits[i].Parent := Panel1;
```

```
    MyEdits[i].Name := 'MyEdit'+IntToStr(i);
```

```
    MyEdits[i].Left := 4;
```

```
    MyEdits[i].Width := self.ClientWidth - 4;
```

```
    MyEdits[i].Height := self.ClientHeight div 8;
```

```
    MyEdits[i].Top := i*MyEdits[i].Height;
```

```
    MyEdits[i].Text := "ABCDEF";
```

```
  end;
```

```
  InputEdit := TInputEdit.Create(self);
```

```
  InputEdit.Parent := Panel1;
```

```
  InputEdit.Left := 4;
```

```
  InputEdit.Color := RGB(0,200,100);
```

```
  InputEdit.Top := 6*MyEdits[4].Height;
```

```
end;
```

```
end.
```

(7) 编译和运行工程 Ex4_4.dpr, 在最下面的文本框中依次输入 ABCDEF, 则在上面的四个文本框中也依次同时出现 ABCDEF, 如图 4.5 所示。

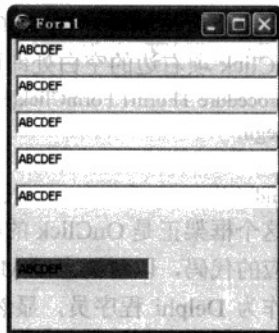


图 4.5 工程 Ex4_4.dpr 的运行结果

当然, 在 WMKeyDown 函数中, 还可以用 Windows 的 API 函数 SendMessage() 和 Postmessage() 来实现消息的转变和发送, 这时 WMKeyDown 函数的代码分别如下 (Unit1.pas 的其他代码不变):

```
procedure TInputEdit.WMKeyDown(var Msg:TMessage); //message WM_KEYDOWN;
```

```
var i:integer;
```

```
begin
```

```
  for i:= 0 to 4 do
```

```
  begin
```

```
    SendMessage(MyEdits[i].Handle,SX_MESSAGE1,Msg.WParam,0);
```

```
  end;
```

```
end;
```

```
procedure TInputEdit.WMKeyDown(var Msg:TMessage); //message WM_KEYDOWN;
```

```
var i:integer;
```

```
begin
```

```
  for i:= 0 to 4 do
```

```
  begin
```

```
    PostMessage(MyEdits[i].Handle,SX_MESSAGE1,Msg.WParam,0);
```

```
  end;
```

```
end;
```

4.4 VCL 事件系统与消息

注意到，每一个 VCL 组件都有若干个事件，所有这些事件组成的系统就是 VCL 事件系统。VCL 事件系统是为程序员能够方便地捕获和处理消息而设置的。一般来说，事件与消息是一一对应的，当产生某个消息时，VCL 就会引发相应的事件并调用其事件处理程序，从而由该事件处理程序完成既定的任务。

例如，在 Delphi 中 OnClick 是用得最多的事件之一，其工作原理是：当单击某一组件（如 TForm）时会产生 Windows 消息 WM_XButtonDown，该消息将引发组件的 OnClick 事件，而 OnClick 事件则调用其事件处理程序，最终完成相应的任务。显然，事件处理程序是程序员的“用武之地”。在 Delphi 的可视化继承开发环境中，利用对象观察器可以非常容易地构造事件处理程序的框架，而无须了解消息是如何引发事件及事件是如何调用其处理程序的。例如，如果需要编写窗体 Form1 的 OnClick 事件处理程序，只需打开对象观察器的 Events 标签页，双击 OnClick 项右边的空白处即可在代码编辑器中创建下列的程序框架：

```
procedure TForm1.FormClick(Sender: TObject);  
begin
```

```
end;
```

这个框架正是 OnClick 的事件处理程序框架，而这时的工作主要是在 begin 和 end 之间编写相应的代码，以对消息 WM_XButtonDown 做出相应的响应。

作为 Delphi 程序员，显然不能满足于“填代码式”的编程模式，而更应该了解和掌握事件响应的基本原理，从深层去编写 Delphi 应用程序，为开发出高效率、高质量的 Delphi 程序奠定基础。为此，先考虑 OnClick 事件的定义方式，其代码如下：

```
TControl = class(TComponent)  
private  
    ...  
    FOnClick: TNotifyEvent;  
    ...  
protected  
    ...  
property OnClick: TNotifyEvent read FOnClick write FOnClick stored IsOnClickStored;  
    ...  
end;
```

此例正是 Delphi 标准控件中 OnClick 事件的定义方式。可以看出，除了 OnClick 被定义为过程类型外，其定义格式与一般属性的直接访问格式几乎完全相同。其中，FOnClick 为事件变量，用于保存事件处理过程的指针，它是由事件名“OnClick”冠以“F”开头而得到的。

从 OnClick 事件的定义方式中可以初步看出事件的基本定义方法。下面的例子将具体说明如何定义一个事件及其处理程序，以及如何将一个消息（包括 Windows 消息及用户自定义消息等）与被定义的事件关联起来，使得产生该消息的时候会引发该事件，进而调用事件处理程序。

例 4.5 本例中定义一个事件 `OnUserEvent` 及其处理程序 `DoUserEvent`，使得鼠标移过 `Panel` 组件上方时引发该事件，并调用该事件处理程序。

注意到，当鼠标移过 `Panel` 组件上方时会产生消息 `WM_MOUSEMOVE`，可以利用该消息处理方法来实现事件的引发，进一步调用事件处理程序。创建程序的步骤如下。

(1) 创建一个新的应用程序，将工程命名为 `Ex4_5.dpr`，主窗体单元文件采用默认的名称 `Unit1.pas`。

(2) 在单元文件 `Unit1.pas` 中，定义类 `TMyPanel`，使之继承类 `TPanel`，其代码如下：

```
TMyPanel = class(TPanel)
private
    { Private declarations }

public
    { Public declarations }

end;
```

(3) 在类 `TMyPanel` 中，先在 `private` 部分声明事件变量 `FOnUserEvent`：

```
FOnUserEvent: TNotifyEvent;
```

`FOnUserEvent` 用于保存事件处理程序的句柄。然后声明事件 `OnUserEvent` 的处理程序 `DoUserEvent`：

```
procedure DoUserEvent(Sender: TObject);
其实现代码如下（在 implementation 部分）：
procedure TMyPanel.DoUserEvent(Sender: TObject);
begin
    showmessage('正在处理.....'); //代表处理代码块
end;
```

(4) 在类 `TMyPanel` 的 `public` 部分中声明属性 `OnUserEvent`：

```
property OnUserEvent: TNotifyEvent read FOnUserEvent write FOnUserEvent;
```

其中，关键字“`read`”表示 `FOnUserEvent` 指示的过程可以对该属性进行读操作，即取值；“`write`”则表示可对该属性进行写操作（如果省略 `write` 部分，则此属性便成为只读），即赋值。读、写方法除了取值和赋值之外，还可以附加其他操作代码，使属性读、写产生附加效应。这正是属性可以取代方法的原因。

(5) 当鼠标移过 `Panel` 组件上方时会产生消息 `WM_MOUSEMOVE`，为使该消息与被定义的事件 `OnUserEvent` 关联起来，必须先捕获该消息，对应的方法声明如下：

```
procedure MouseOverPanel(var Msg: TMessage); message WM_MOUSEMOVE;
该方法是在类 TMyPanel 的 private 部分中声明，其实现代码如下（在 implementation 部分）：
procedure TMyPanel.MouseOverPanel(var Msg: TMessage);
begin
    MyPanel.OnUserEvent := DoUserEvent; //使事件 OnUserEvent 与其处理程序关联起来
    FOnUserEvent(self);                 //调用事件处理程序 DoUserEvent
end;
```

这时类 `TMyPanel` 的完成代码如下：

```
TMyPanel = class(TPanel)
```



```

private
{ Private declarations }
FOnUserEvent: TNotifyEvent;
procedure MouseOverPanel(var Msg:TMessage); message WM_MOUSEMOVE;
procedure DoUserEvent(Sender: TObject);
public
{ Public declarations }
property OnUserEvent: TNotifyEvent read FOnUserEvent write FOnUserEvent;
end;

```

(6) 在单元 Unit1.pas 的 interface 部分声明类 TMyPanel 的实例 MyPanel:

```

var
Form1: TForm1;
MyPanel: TMyPanel;

```

然后在窗体的 OnCreate 事件处理程序中创建实例对象 MyPanel, 并显示, 相应代码如下:

```

procedure TForm1.FormCreate(Sender: TObject);
begin
MyPanel := TMyPanel.Create(self);
MyPanel.Parent := self;
MyPanel.Width := 2*self.ClientWidth div 3;
MyPanel.Height := 2*self.ClientHeight div 3;;
MyPanel.Color := RGB(0,255,255);
MyPanel.Left := self.ClientWidth div 6;
MyPanel.Top := self.ClientHeight div 7;
end;

```

(7) 编译、运行该程序, 结果如图 4.6 所示。当鼠标移过组件 Panel 上方时, 事件 OnUserEvent 被触发, 同时 OnUserEvent 调用其事件处理程序 DoUserEvent, 弹出“正在处理.....”的消息框。

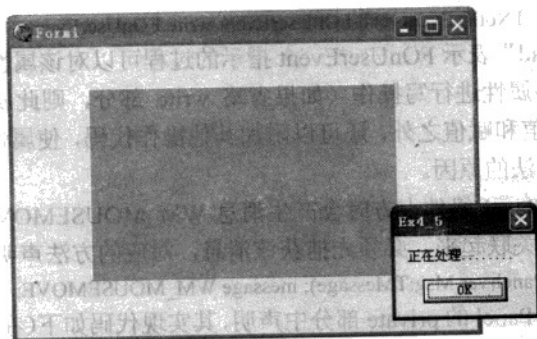


图 4.6 程序 Ex4_5.dpr 的运行界面

4.5 VCL 消息处理机制

在介绍 VCL 消息的处理机制之前不得不先了解类 TApplication。

类 TApplication 继承类 TComponent, 在单元 Forms 中声明。其方法和属性包括了 Windows

操作系统中创建、运行和销毁应用程序等既定的基本操作和属性,因此在用 Delphi 编写 Windows 应用程序时简化了用户和 Windows 环境之间的接口,大多数程序都只是包含三条语句:

```
Application.Initialize;  
Application.CreateForm(TForm1, Form1);  
Application.Run;
```

类 TApplication 封装的功能主要分为四类:Windows 消息处理;菜单加速和键盘处理;异常处理;上下文联机帮助。显然,在此涉及的是“Windows 消息处理”这一功能,处理的过程由 TApplication 类对象的 Run 方法开始。

Run 方法的作用是启动消息循环,然后通过 Application.HandleMessage 方法调用 Application.ProcessMessage 方法,该函数会在应用程序的消息队列中检索消息。当有消息被检索到时,则触发 Application.OnMessage 事件,即当应用程序收到消息时会产生该事件,其定义如下:

```
type TMessageEvent = procedure (var Msg: TMsg; var Handled: Boolean) of object;  
property OnMessage: TMessageEvent
```

OnMessage 事件只接收登记的消息,它优于任何消息处理,其相应的事件处理程序必须是 TMessageEvent 类型,声明如下:

```
type TMessageEvent = procedure (var Msg: TMsg; var Handled: Boolean) of object;
```

其中 TMsg 是 Windows 中定义的消息记录。当事件处理程序定义完了以后就可以赋给 OnMessage 事件,例如:

```
Procedure OnMyMessageHandled (var Msg:TMsg;var Handled:Boolean);  
... ..  
Application.OnMessage := OnMyMessageHandled;
```

在 Delphi 应用程序运行过程中,Run 方法不断循环调用 HandleMessage 方法,直到 Application.Terminate 值为 true 时才停止应用程序的运行。不断调用 HandleMessage 方法的过程实际上就是不断调用 ProcessMessage 方法对系统消息队列进行消息检索的过程。在这一过程中,一旦有消息被捕获将立即判断自己的 OnMessage 事件处理程序句柄,调用相应的事件处理程序,以完成相应的任务。由于 SendMessage 和 Perform 方法所发送的消息不会进入系统消息队列,因此 ProcessMessage 方法无法捕获到这些消息,也就无法对这些消息进行响应。

回顾前面可知,它可以通过利用 OnMessage 事件来捕获消息并对其进行处理和响应,这是在 Delphi 应用程序中第一个可以截获和处理消息的地方。

对于被 ProcessMessage 方法捕获到的消息,它将被 API 函数 DispatchMessage 分发给 StdWndProc 函数。StdWndProc 函数是一个汇编函数,它的作用是把接收到的消息发送给 VCL 对象。VCL 对象用于接收消息的方法称为 MainWndProc,它是 TWinControl 类中的静态方法,不能被重载,且它不直接处理消息。

当消息离开 MainWndProc 后,消息被传递给对象的 WndProc 方法。WndProc 方法是在 TControl 类中定义的一个虚拟方法,它可以在 TControl 类的派生类中通过对 WndProc 进行覆盖的方法来处理消息。这是在 Delphi 应用程序中第二个可以截获和处理消息的地方。

当离开 WndProc 方法后,消息将由 Dispatch 方法进行分发。Dispatch 分发消息的原则是,根据传入的 Message 来寻找相应的处理方法,以把消息分送给已经定义或已经存在于 VCL 对

象之中的特定的消息处理方法。如果找到则调用该方法，否则回溯到对象的父类中进行查找，一直回溯到类 TObject 为止。而类 TObject 则提供了默认的消息处理方法——DefaultHandler 方法，即如果一个消息一直没有找到合适的消息处理方法，最终将由 DefaultHandler 方法进行处理。DefaultHandler 方法对消息进行最后的处理，然后把消息传递给 Windows 的 DefWindowProc 函数或其他默认的窗口过程。

4.6 小结

要写出高质量的 Delphi 应用程序，必须对它的运行机制及其与 Windows 系统之间的关系和“往来”有较为深刻的理解。本章主要介绍 Windows 消息的结构特征及其驱动机制、Delphi 应用程序捕获和处理 Windows 消息的途径和方法，以及几种不同类型的消息和它们的传递方法等，同时对各个问题的阐述都给出了具体的实例说明。通过对本章的学习，可以加深对 Delphi 程序运行机制的理解和掌握，为 Delphi 应用编程奠定基础。学完本章后应掌握下列知识点：

- Windows 消息的记录结构及其驱动机制，包括消息在队列中等待和循环方法等。
- 在 Delphi 应用程序中捕获消息，然后对其处理的一般编程方法。
- Delphi 消息系统中 VCL 内部消息、通知消息、自定义消息的基本特征及其传递方法。
- 消息处理的一般机制。

第 5 章 GUI 应用程序开发

GUI 应用程序开发是当今应用软件开发的主流方向, Delphi 为此提供了完美的支持。可以说, Delphi 应用程序设计的过程就是 GUI 应用程序开发的过程。在 Delphi for .NET 应用程序开发中, VCL 组件的使用是 VCL Forms 应用程序界面设计的基础。熟悉常用组件的属性和方法, 对可视化程序设计是非常有帮助的; 熟悉单文档窗体和多文档窗体的开发, 以及 Delphi 应用程序菜单设计也是 GUI 应用程序开发的基础。本章涉及的要点主要包括:

- 常用 VCL 组件的使用方法和技巧。
- 单文档窗体和多文档窗体的开发。
- Delphi 应用程序菜单设计方法。
- GUI 应用程序开发实例。

5.1 关于 GUI 应用程序

GUI 是英文句子“Graphic User's Interface”的简写, 译为中文就是“图形用户界面(接口)”。图形用户界面技术的产生是操作系统的一次革命性更替, 现已渗透到各个领域当中, 在很多方面得到了成功的应用。其最大的成功之处在于其设计的人性化, 可以说这是操作系统设计上的的一大突破, 已经形成了事实上的界面标准。目前, 大多数程序设计都采用 GUI 技术, Delphi 更是堪称这方面的楷模。

在 Delphi 的集成开发环境 (IDE) 中, 利用窗体设计器与有关组件的搭配使用可以设计出非常精美、功能强大的应用程序——GUI 应用程序。因此, 作为 Delphi 程序员, 熟练掌握常用组件、窗体设计器的有关方法 (事件) 和属性及单文档和多文档程序的设计方法等, 对于快速、高效开发 GUI 应用程序是至关重要的。本章将结合大量的具体实例就如何运用组件和窗体设计器等开发具有相应功能的 GUI 应用程序进行说明。此前, 需要先了解有关组件和窗体的基本知识。

组件 (Component) 是 GUI 应用程序编程的“元件”, 是可视化编程的基础。它是一种虚拟的对象, 在设计阶段设置和使用, 其源于 TComponent。应用程序的界面几乎都是由组件构成的, 程序的功能也是通过对组件属性设置和动态修改来完成的。对于程序员来说, 组件的使用非常直观, 只要从工具面板上选择一些组件放到窗体, 然后进行窗体设计, 就可以在较短的时间设计出精美的界面。因此, 了解组件、掌握组件的运用是 Delphi 集成可视化开发环境中进行应用程序设计和开发的前提和保证。

经过多年的努力, 人们已经开发出大量的组件, 形成了称为 VCL (Visual Component Library) 的组件库。Delphi 2005 开发环境中的应用程序开发更是离不开这些组件。Delphi 2005 VCL 中的组件可以分为四类, 即标准组件、Windows 组件、非可视组件和自定义组件。

1. 标准组件

标准组件是指 Delphi 本身提供的标准的组件, 包括数据库组件、网络组件、图形组件等。

2. Windows 组件

Windows 组件是 Delphi 利用 Windows 公共控件进行开发而得到的组件, 实际是 Windows 控件的 Delphi 包装。TRichEdit 组件、TListView 组件等都是 Windows 组件。

3. 非可视组件

非可视组件是指那些在程序运行过程中看不到的组件 (当然, 在程序设计过程中是可以看到的)。利用这些组件的目的就是利用它提供的功能, 因为这些组件在对象内封装了实体的功能, 可以提供大量接口函数, 可轻易地对它编程。TTimer 组件、TTable 组件等都是典型的非可视组件。

4. 自定义组件

自定义组件是指由程序员根据任务特性自己开发的组件, 这些组件不在 VCL 中。每个软件开发项目都有自己的特点, 大多都有重复的工作, 可把一些重复的代码编写工作设计成组件 (VCL 中没有的), 以减少以后的工作量, 降低开发成本。

Delphi 中的窗体来源于 Windows 系统中的窗口结构。Windows 窗体已形成事实上的窗体标准, 很多软件公司都以此为参照。Windows 窗体主要包含标题、菜单、快捷方式工具栏及主窗口等。Delphi 也不例外, 其窗体也主要包含标题、菜单、快捷方式工具栏及主窗口等部分, 其中, 标题栏上通常有三个按钮: 关闭按钮、最大化按钮和最小化按钮及标题标识符等。

Delphi 窗体可以分为两种类型, 即单文档窗体和多文档窗体。与 Windows 系统中的窗体类似, 在 Delphi 中每个窗体都有一个惟一的标识符, 称为窗体的句柄 (Handle), 它是窗口的标志, 调用一个窗口都要通过其句柄来完成。

对程序员而言, 窗体是一个开发和设计的工作环境; 而对用户而言, 窗体是一个软件的界面。该界面直接影响到用户对软件的喜爱程度及其对软件的评价。因此设计一个良好的界面是 GUI 应用程序设计的主要目标之一, 设计的好坏有赖于开发者的经验和其审美观。一般来说, 要注意以下几点。

(1) 界面友好、简洁。简洁的界面符合人的从简习惯 (几乎没有人说界面越复杂越好), 友好的界面会增强人的亲切感、激起人的兴趣。因此, 在设计窗体时应尽可能做得简单而不单调, 亲切而不令人厌烦。

(2) 界面的一致性。一致性是指界面的统一性、协调性, 这包括窗口元素大小、颜色、位置等的统一性和协调性。不一致的界面是令人厌恶的。

(3) 界面的标准性。标准性是指符合 Windows 系统窗体界面的标准。因为现在绝大部分的用户都习惯了 Windows 系统窗体界面, 如果设计了与 Windows 系统窗体界面另类的界面, 那将会令用户觉得非常别扭, 需要长时间来适应, 甚至需要大量的资本来培训, 是不受欢迎的。

(4) 突出界面的重点。在实际中, 工作往往是重复性的, 而重复的工作一般都是重点的、操作频繁的工作。因此, 在用软件实现时, 软件界面应该能突出完成这些工作的模块, 使得用户对这些经常用的操作可以“随手可得”。这部分功能通常做成快捷方式, 或者放在比较明显的地方。

总之, 软件开发的过程是一个经验不断积累的过程, 良好的界面不但有赖于开发者的灵感和审美观, 还依赖于开发者不断积累的经验和技术。所以希望读者能对 GUI 应用程序设计的各个“元件”都能熟练掌握, 包括熟练掌握常用组件及窗体等的有关属性和方法等, 为能设计一个出色的 GUI 应用程序奠定坚实的基础。

5.2 按钮组件

5.2.1 Button 组件

Button 组件几乎在每一个 Windows 应用程序中都会用到,它允许用户通过单击来执行某些操作。单击一个按钮相当于执行相应的一个函数,所以其逻辑性比较强。在工具面板中,它位于 Standard 标签页上。Button 组件的许多属性和方法是大多数组件所共有的,所以下面着重介绍该组件,读者可以由此学习其他组件与之共有的属性和方法。

1. Button 组件常用的属性

当在窗体中选中了 Button 组件以后,对象观察器中就会列出 Button 组件所有的属性和方法。这些属性和方法主要包括以下几种:

(1) Name 属性。Name 属性是每个组件都有的,是组件的惟一标识,只能在程序设计时在对象观察器中设定和更改,在程序运行时不能更改。对于 Button 组件,Delphi 默认设定为 Button1 (如果是窗体上放的第二个按钮则默认值为 Button2,第三个为 Button3,其他组件也有类似的情况)。

在对象观察器 (Inspector) 中,展开 Miscellaneous 标签就可以看到 Name 属性项。

熟悉各种属性在对象观察器中的位置,这对于提高编程速度非常重要。另外,同一个属性可能在多个地方出现。

(2) Caption 属性。Caption 属性用于设置按钮上显示的文本。例如,如果把 Caption 属性值设为“确认”,则按钮将显示“确认”这两个字。如果在文本中某个字符加上符号“&”,则表示该字符是按钮的快捷方式。例如,把 Caption 属性值设为“确认(&P)”,则按钮上将显示文本“确认(P)”,这时按键盘上的 P 键就相当于用鼠标单击该按钮 (程序运行时),P 键为按钮的快捷方式。

在程序中,可以用下列代码动态更改 Caption 属性值:

```
Button1.Caption := '更改的文字';
```

在对象观察器中,展开 Action 标签就可以看到该属性项。

(3) Enabled 属性。Enabled 属性用于设置组件的可用性,当 Enabled 属性值为 true 时组件可用;当 Enabled 属性值为 false 时组件不可用,这时按钮就会变成灰色,不会响应用户对它的操作。但是不管 Enabled 属性取什么值,在程序中都可以动态改变它。例如:

```
Button1.Enabled := true;
```

在对象观察器中,展开 Visual 标签或者 Input 标签就可以看到该属性项。

(4) Visible 属性。Visible 属性值设为 true (默认值) 时组件可见,如果为 false 时组件则不可见。其编程方式为:

```
Button1.Visible:=false;
```

该属性位于 Action 标签页上。

(5) Size 属性。Size 属性用于设置按钮上文本字体的大小,如果设为 30,那么按钮上文本就以 30 号字显示。其编程方式为:

```
Button1.Font.Size :=30;
```

该属性输入 Font 属性的子属性, 位于 Visual 标签页上。

(6) Hint 属性和 ShowHint 属性。在 Windows 应用程序中经常会遇到, 当把鼠标放在某一组件上停留片刻时, 提示信息框自动弹出来, 显示相应的帮助信息。在 Delphi 2005 中就是通过对属性 Hint 和 ShowHint 的设置来完成的。其中 Hint 属性值就是弹出的信息, 而 ShowHint 属性值为 true 时信息才能被弹出; 如果为 false, 即使设置了 Hint 属性, 信息也不会被弹出。例如, 要使得鼠标停留在 Button1 上时弹出“这是退出按钮”, 那么这两项就应该分别设置为: Hint 属性值为“这是退出按钮”, ShowHint 属性值为 true。

这两个属性位于 Help and Hints 标签页上。

(7) Cursor 属性。该属性用于设置当鼠标移动组件上面时的形状, 它位于 Visual 标签页上。在找到该属性后, 单击其右边的下列按钮, 就会出现如图 5.1 所示的下拉框, 从中选择相应的鼠标形状即可。

2. Button 组件常用的方法 (或事件)

图 5.2 列出了 Button 组件所有的方法, 下面对一些常用的方法做一下简要说明。

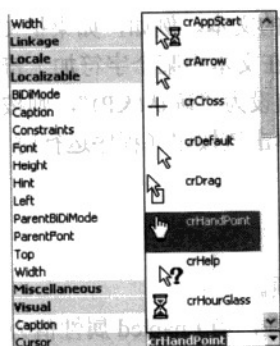


图 5.1 Cursor 属性值的设置

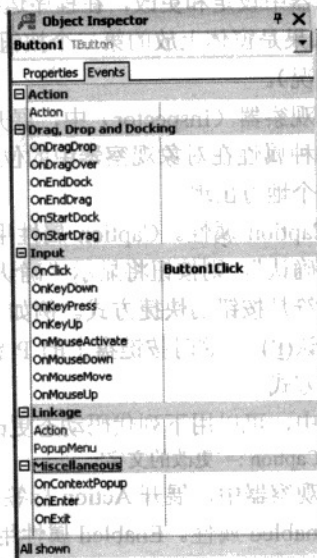


图 5.2 Button 组件的方法

(1) OnClick 方法。该方法是最常用的方法。当鼠标在 Button 按钮上单击时该事件被触发。

在窗体上双击 Button 组件, 则 Delphi 会自动生成被触发事件的处理过程, 并且鼠标会自动移动该过程的代码编辑处, 同时会自动地把生成的过程名称设置为 OnClick 项的值。当然, 也可以通过改变 OnClick 项的值的方法来更改相应过程的名称。例如, 双击窗体上的 Button1, 则 Delphi 自动生成的过程如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
end;
```

上述的过程名称的修改也只能通过修改 OnClick 项的值来完成, 而不能直接在代码编辑器

中修改。

除了 OnClick 外, Button 组件还有其他许多方法。但除了触发方式不同以外,其他方面都一样。因此,下面主要介绍相关方法的触发条件。

(2) OnKeyDown 方法。当按下键盘上某个键时,该方法被触发。

(3) OnKeyPress 方法。当按下键盘上某个 ASCII 码键时,该方法被触发。

(4) OnKeyUp 方法。当松开键盘上某个键时,该方法被触发。

(5) OnMouseDown 方法。当对 Button 组件单击鼠标时,该方法被触发,这时 OnClick 方法将失效。

(6) OnMouseMove 方法。当移动鼠标过 Button 组件上面时,该方法被触发。

(7) OnMouseUp 方法。当对 Button 组件松开鼠标键时(当然先按下过),该方法被触发。

5.2.2 BitBtn 组件

BitBtn 组件(位图按钮)也是一个 Button 按钮对象,它位于工具面板上的 Additional 标签中。其功能和作用与 Button 组件基本相同,所不同的是,在 BitBtn 组件上可以添加图标,使界面看起来更加美观。

除了与 Button 组件有大部分相同的属性外, BitBtn 组件还具有自己特有的属性(两者的方法相同),以下作简要介绍。

1. Kind 属性

Kind 属性用于设置按钮上的图标,该属性位于 Miscellaneous 标签页上。

在 Kind 属性项的右边有一个下拉列表框,可选项包括 bkAbort 等 11 个值。选择不同的值将得到不同的位图,其对应的位图分别如表 5.1 所示。

表 5.1 Kind 属性值与位图的对应关系

Kind 属性值	位图按钮的效果
bkAbort	
bkAll	
bkCancel	
bkClose	
bkCustom	用于自定义位图
bkHelp	
bkIgnore	
bkNo	
bkOk	
bkRetry	
bkYes	

设置 Kind 属性值以后,按钮上位图的右边都有文字说明。如果觉得这些文字不合理,则可通过修改 Caption 属性值来改变它们。另外,当 Kind 属性取默认值 bkCustom 时,按钮上没

有位图。这时可以利用 Glyph 属性来加载位自定义的图文件。

2. Glyph 属性

Glyph 属性的右边有一个下拉列表框, 单击它将弹出 Picture Editor 对话框, 如图 5.3 所示。

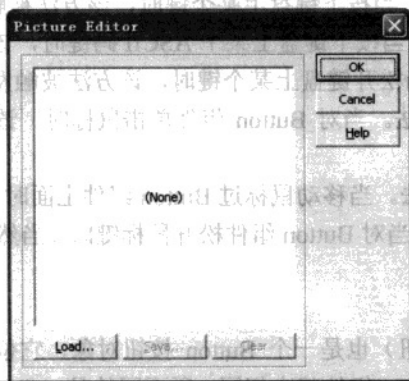


图 5.3 Picture Editor 对话框

单击 Load... 按钮, 就可以加载已制作好的位图文件。

提示: 位图文件可以用 Windows 系统的画图程序来制作, 也可以用 Delphi 2005 IDE 提供的工具来完成, 即: 选择 IDE 菜单命令 Tools → Image Editor, 将打开图像编辑器, 在此可以制作自己喜欢的位图。

5.2.3 CheckBox 组件与实例

CheckBox 组件 ☒ (复选框组件) 位于工具面板上的 Standard 标签中。它主要用来决定是否选择某项内容, 常与 GroupBox 组件结合使用。其许多的属性和方法都与 Button 组件类似, 在此不赘述, 而只介绍其特有的一些属性和方法。

1. Checked 属性

Checked 属性是 CheckBox 组件最重要的一个属性, 其值是布尔型。当 CheckBox 组件被选中时 Checked 属性值为 true, 如果未被选中则 Checked 属性值为 false。反过来, 如果 Checked 属性值被设置为 true, 则 CheckBox 组件显示被选中状态; 如果被设置为 false, 则显示未被选中状态。例如, 如果下列语句被执行, 那么 CheckBox1 组件将显示被选中状态:

```
Checkbox1.Checked := true;
```

Checked 属性位于 Action 标签页上。

2. Alignment 属性

Alignment 属性 (位于 Visual 标签页上) 有两个值——taRightJustify 和 taLeftJustify。如果 Alignment 属性值为 taRightJustify, 则标签位于框的右边, 如图 5.4(a) 所示; 如果为 taLeftJustify, 则标签位于框的左边, 如图 5.4(b) 所示。

3. AllowGrayed 属性

AllowGrayed 属性其值为布尔型, 当它取值为 true 时允许 CheckBox 组件有三个状态, 即选中、不选中、选中但变灰。这时, 当不断地单击该组件, 则组件状态将在这三个状态中变动。

AllowGrayed 属性为 false, 表示不允许三种状态, 只能有选中、不选中两种状态。AllowGrayed 属性位于 Miscellaneous 标签页上。

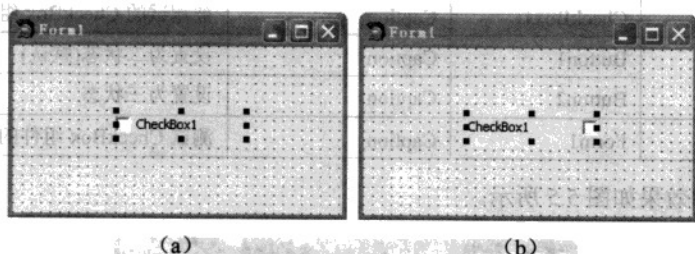


图 5.4 Alignment 属性的取值与标签位置的关系

4. State 属性

该属性用来设置或返回 CheckBox 组件的状态, 当 State 属性为 cbChecked 时 CheckBox 组件显示选中状态, 为 cbUnchecked 时则显示未被选中状态, 为 cbGrayed 时则显示被选中但变灰状态 (不确定状态)。显然, 如果 AllowGrayed 属性值为 true, 则 State 属性值可以返回三种状态, 否则只能返回两种状态, 即 cbChecked 和 cbUnchecked。

5. OnClick 方法

OnClick 方法也是 CheckBox 组件最常用的方法。当单击 CheckBox 组件时该方法被触发, 同时 CheckBox 组件也会发生改变。在程序设计时, 只要双击该组件即可进入 OnClick 方法的代码编辑处, 这时产生的函数名是默认的, 如果需要更改, 可在对象观察器选择事件选项卡, 然后在 OnClick 项的右边文本框中更改。

从上面的介绍可以看到, CheckBox 组件的状态由两个属性值来体现: Checked 属性值和 State 属性值。这两个属性有什么不同呢, 又有什么关系呢? 下面通过一个测试 CheckBox 组件状态的程序来理解, 同时也加深对 CheckBox 组件 OnClick 方法的理解和使用。

例 5.1 CheckBox 组件的 Checked 属性和 State 属性的测试和运用。

构造测试 CheckBox 组件和运用 CheckBox 组件的示范程序可以通过下列步骤来完成。

(1) 创建一个 VCL 应用程序, 工程名称设为 Ex5_1.dpr, 窗体单元文件名取默认值 Unit1.pas, 保存在文件夹 Ex5_1 中。

(2) 在窗体上添加两个 Label 组件、两个 Edit 组件、一个 CheckBox 组件和两个 Button 组件 (按钮), 然后设置各组件的属性, 如表 5.2 所示。

表 5.2 窗体及组件属性设置列表

组件类型	组件名称	属性设置项目	设置结果
Label	Label1	Caption	Checked 属性的当前值:
		Size	13
	Label2	Caption	State 属性的当前值:
		Size	13
Edit	Edit1	Text	(空值)
	Edit2	Text	(空值)

续表

组件类型	组件名称	属性设置项目	设置结果
CheckBox	CheckBox1	Caption	被测试的 CheckBox 组件
Button	Button1	Caption	设置为二状态(默认)
	Button2	Caption	设置为三状态
Form	Form1	Caption	测试 CheckBox 组件的状态

窗体设计的效果如图 5.5 所示。

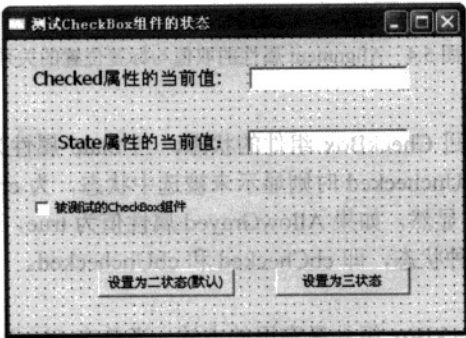


图 5.5 窗体设计的效果 (工程 Ex5_1.dpr)

(3) 添加代码。在窗体中双击 CheckBox 组件，则进入 OnClick 方法的代码编辑处，并添加相应的代码。代码的功能是，当单击 CheckBox 组件时获取它的状态，并将状态值在两个编辑框中显示。代码如下：

```
procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    if CheckBox1.Checked then //在 Edit1 编辑框中显示 Checked 属性值
        Edit1.Text := 'true'
    else
        Edit1.Text := 'false';

    if CheckBox1.State = cbChecked then //在 Edit2 编辑框中显示 State 属性值
        Edit2.Text := '被选中'
    else if CheckBox1.State = cbGrayed then
        Edit2.Text := '被选中但变灰'
    else
        Edit2.Text := '未被选中';
end;
```

Button1 和 Button2 组件的功能分别是把 CheckBox 组件的状态设置为两种(默认是两种状态)和三种，实际上就是设置 CheckBox 组件的 AllowGrayed 属性值。这些功能的完成是利用了 Button 组件的 OnClick 方法，代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
    CheckBox1.AllowGrayed := false;    //设置为两种状态
    Button1.Enabled := false;          //设置完后, 自己变无效
    Button2.Enabled := true;           // Button2 变有效
end;
```

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    CheckBox1.AllowGrayed := true;     //设置为三种状态
    Button2.Enabled := false;          //设置完后, 自己变无效
    Button1.Enabled := true;           // Button1 变有效
end;
```

另外, 为了在程序刚启动完毕时界面也能显示 CheckBox 组件的状态 (初始化), 需要在窗体的 FormCreate 方法中加入一些代码, 结果如下:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    if CheckBox1.Checked then //在 Edit1 编辑框中显示 Checked 属性值
        Edit1.Text := 'true'
    else
        Edit1.Text := 'false';
    if CheckBox1.State = cbChecked then //在 Edit2 编辑框中显示 State 属性值
        Edit2.Text := '被选中'
    else if CheckBox1.State = cbGrayed then
        Edit2.Text := '被选中但变灰'
    else
        Edit2.Text := '未被选中';
    Button1.Enabled := false;
end;
```

代码编写完成以后, 整个单元文件的代码如下:

```
unit Unit1;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, StdCtrls;

type
    TForm1 = class(TForm)
        Button1: TButton;
        CheckBox1: TCheckBox;
        Label1: TLabel;
        Label2: TLabel;
        Edit1: TEdit;
        Edit2: TEdit;
        Button2: TButton;
        procedure FormCreate(Sender: TObject);
```

```
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    if CheckBox1.Checked then
        Edit1.Text := 'true'
    else
        Edit1.Text := 'false';
    if CheckBox1.State = cbChecked then
        Edit2.Text := '被选中'
    else if CheckBox1.State = cbGrayed then
        Edit2.Text := '被选中但变灰'
    else
        Edit2.Text := '未被选中';
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    CheckBox1.AllowGrayed := false;
    Button1.Enabled := false;
    Button2.Enabled := true;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    CheckBox1.AllowGrayed := true;
    Button1.Enabled := true;
    Button2.Enabled := false;
end;

procedure TForm1.FormCreate(Sender: TObject);
```

```
begin
```

```
  if CheckBox1.Checked then
```

```
    Edit1.Text := 'true'
```

```
  else
```

```
    Edit1.Text := 'false';
```

```
  if CheckBox1.State = cbChecked then
```

```
    Edit2.Text := '被选中'
```

```
  else if CheckBox1.State = cbGrayed then
```

```
    Edit2.Text := '被选中但变灰'
```

```
  else
```


```
    Edit2.Text := '未被选中';
```

```
  Button1.Enabled := false;
```

```
end;
```

```
end.
```

(4) 编译、运行程序。

单击工具栏上的快捷按钮  (Run)，如果编译没有错误，将出现如图 5.6 所示的运行界面。

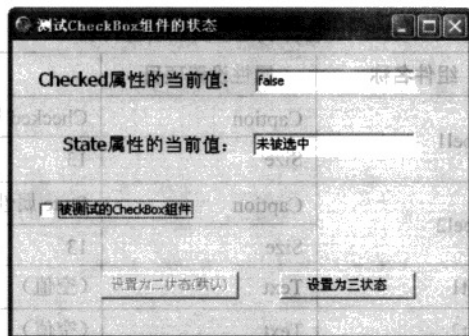


图 5.6 工程 Ex5_1.dpr 的运行界面

不断地单击 CheckBox 组件，可以看到编辑框中数据的变化；这种测试可以在二状态和三状态之间切换进行，综合分析 Checked 属性值（布尔型，二状态）与 State 属性值（二或三状态）的关系。从观察结果可以推出它们之间的关系，如表 5.3 所示。

表 5.3 Checked 属性值与 State 属性值的关系

Checked 属性值	State 属性值	备注
True	cbChecked	被选中
False	cbGrayed	被选中但变灰
	cbUnchecked	未被选中

当 CheckBox 组件的 AllowGrayed 属性值为 true 时，State 属性值才有三个状态，否则只有两个状态（默认值）。

5.2.4 RadioButton 组件与实例

RadioButton 组件^①通常称为单选按钮组件，它位于工具面板上的 Standard 标签中。它与 CheckBox 组件有类似的功能，不同的是，当多个组件放在一个 Form、Panel 或 GroupBox 等同一容器中而组成一组时，它们彼此是相互关联的（而 CheckBox 组件则没有关联）。充分利用这种关联性可以使得编程变得简洁。

RadioButton 组件之间的关联性体现在：在一个容器中有多个 RadioButton 组件时，只能选中一个 RadioButton 组件，如果选中其他的 RadioButton 组件，则当前被选中的 RadioButton 组件自动变为不被选中状态。

应该说，除了 RadioButton 组件之间彼此有关联外，在其他方面它与 CheckBox 组件的主要特性基本一样，如 Checked 属性、Alignment 属性、OnClick 方法等都是同样的；但也有不同，如 RadioButton 组件没有 AllowGrayed 属性和 State 属性等。

例 5.2 RadioButton 组件的运用范例。

下面用两个 RadioButton 组件和一个 GroupBox 组件来代替工程 Ex5_1（见例 5.1）中的两个 Button 组件，得到的工程命名为 Ex5_2.dpr，窗体单元文件名为 Unit1.pas。各组件属性的设置情况如表 5.4 所示。

表 5.4 窗体及组件属性设置列表

组件类型	组件名称	属性设置项目	设置结果
Label	Label1	Caption	Checked 属性的当前值：
		Size	13
	Label2	Caption	State 属性的当前值：
		Size	13
Edit	Edit1	Text	（空值）
	Edit2	Text	（空值）
CheckBox	CheckBox1	Caption	被测试的 CheckBox 组件
GroupBox	GroupBox1	Caption	设置 CheckBox 状态
RadioButton	RadioButton1	Caption	二状态
	RadioButton1	Caption	三状态
Form	Form1	Caption	测试 CheckBox 组件的状态

设计效果如图 5.7 所示。

然后在 TForm1.FormCreate 函数中添加下列语句，用于把 RadioButton1 设置为被选中状态。

```
RadioButton1.Checked := true;
```

最后分别在 RadioButton1 和 RadioButton2 的 OnClick 方法中添加下列语句：

```
CheckBox1.AllowGrayed := false; //把 CheckBox1 设置为二个状态
```

和

```
CheckBox1.AllowGrayed := true; //把 CheckBox1 设置为三个状态
```

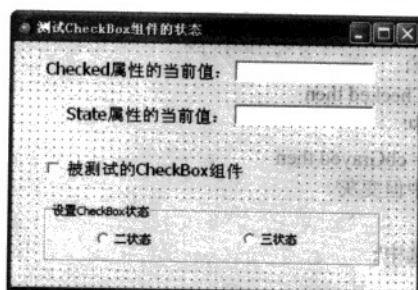


图 5.7 窗体设计的效果 (工程 Ex5_2.dpr)

至此，代码修改工作完成，单元文件的代码如下。

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    CheckBox1: TCheckBox;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    GroupBox1: TGroupBox;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    procedure RadioButton2Click(Sender: TObject);
    procedure RadioButton1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
  if CheckBox1.Checked then
    Edit1.Text := 'true'
```

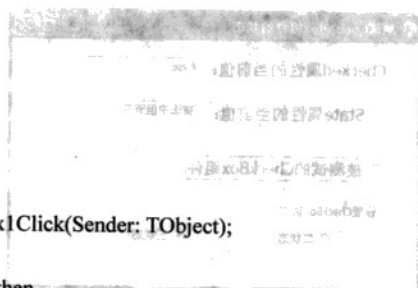


图 5.8


```

else
    Edit1.Text := 'false';
if CheckBox1.State = cbChecked then
    Edit2.Text := '被选中'
else if CheckBox1.State = cbGrayed then
    Edit2.Text := '被选中但变灰'
else
    Edit2.Text := '未被选中';
end;

```

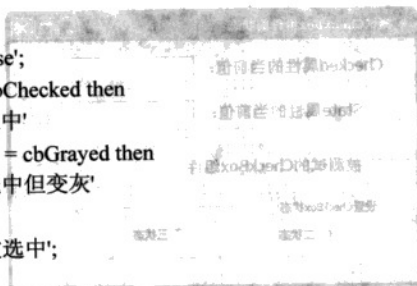


图 5.7 测试CheckBox组件的状态

```

procedure TForm1.FormCreate(Sender: TObject);
begin

```

```

    if CheckBox1.Checked then
        Edit1.Text := 'true'
    else
        Edit1.Text := 'false';
    if CheckBox1.State = cbChecked then
        Edit2.Text := '被选中'
    else if CheckBox1.State = cbGrayed then
        Edit2.Text := '被选中但变灰'
    else
        Edit2.Text := '未被选中';
    RadioButton1.Checked := true;
end;

```

```

procedure TForm1.RadioButton1Click(Sender: TObject);
begin

```

```

    CheckBox1.AllowGrayed := false;
end;

```

```

procedure TForm1.RadioButton2Click(Sender: TObject);
begin

```

```

    CheckBox1.AllowGrayed := true;
end;
end.

```

运行修改后的程序结果如图 5.8 所示。

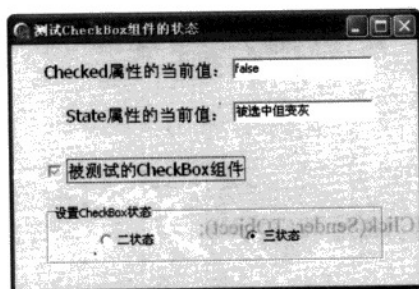


图 5.8 工程 Ex5_2.dpr 的运行界面

程序运行结果与修改前的结果完全一致。从这个程序中可以深入领会到 `RadioButton` 组件的基本使用方法。

5.2.5 SpeedButton 组件

`SpeedButton` 组件通常为加速按钮或彩色按钮，其功能（方法）与 `Button` 组件一样。不同的是，它的外观可以做得很漂亮，按钮上可显示图片和文本。它位于 `Additional` 标签页上。

5.3 文本组件

文本组件是用于显示文本和编辑文本的组件，主要包括 `Label` 组件、`Memo` 组件、`Edit` 组件等。

5.3.1 Label 组件

`Label` 组件（通常称为标签）用于显示文本，但不能编辑文本，位于工具面板的 `Standard` 标签页上。前面的许多程序都已经用到了 `Label` 组件。下面对其常用的属性和方法作进一步介绍。

1. Caption 属性

该属性是 `Label` 组件最重要、最常用的属性，与 `Button` 组件的 `Caption` 属性一样，用于设置组件上面显示的按钮。

在程序中，可以用下列代码动态更改 `Caption` 属性值：

```
Button1.Caption := '显示的文字';
```

这可以使组件显示的文本随响应运行事件而作出相应的更改。

在对象观察器中，展开 `Action` 标签就可以看到该属性项。

2. AutoSize 属性

`AutoSize` 属性值是布尔型，位于 `Visual` 标签页上。当它的值为 `true` 时（默认值）`Label` 标签将根据其中文本的大小、自动标签的宽（`width`）和高（`height`），使之与文本的宽和高一样；当为 `false` 时 `Label` 标签的宽（`width`）和高（`height`）将由用户定义，但只有在 `BorderStyle` 属性值为 `Single` 时才有效。

3. Layout 属性和 Alignment 属性

`Layout` 属性用于设置文本在垂直方向上的位置，位于 `Miscellaneous` 标签页上。它包含三个值，即 `tlBottom`、`tlCenter` 和 `tlTop`，其中，当 `Layout` 属性值为 `tlCenter` 时文本将居于 `Label` 标签的中间（垂直方向上），为 `tlBottom` 时将居于 `Label` 标签的底部，为 `tlTop` 时将居于 `Label` 标签的顶部。可以用下列语句动态改变文本在垂直方向上的位置（前提是 `AutoSize` 属性值为 `false`）：

```
label1.Layout := tlBottom; //tlTop 或 tlCenter
```

`Alignment` 属性用于设置文本在水平方向上的位置，位于 `Visual` 标签页上。它也包含三个值，即 `taCenter`、`taLeftJustify` 和 `taRightJustify`，其中，当 `Alignment` 属性值为 `taRightJustify` 时文本将跟 `Label` 标签右边对齐，为 `taLeftJustify` 时将跟 `Label` 标签左边对齐，为 `taCenter` 居于 `Label` 标签的中间（水平方向上）。可以用下列语句动态改变文本在水平方向上的位置（前提是 `AutoSize` 属性值为 `false`）：

```
label1.Alignment := taRightJustify; //taLeftJustify 或 taCenter
```

4. Width 属性和 Height 属性

Width 属性值和 Height 属性值分别用于设置 Label 标签的宽和高,单位为像素。当 AutoSize 属性值为 false 时,这两个属性才有效。Width 属性值也可以用下列语句动态改变 (Height 属性值则不行):

```
label1.Width := 100;
```

因此基于这样的动态特性可以利用 Label 组件制作进程条等。

5. Anchors 属性

该属性位于 Layout 标签下,有四个子属性,即 akLeft、akTop、akRight、akBottom,它们都是布尔型,其含义见表 5.5。

表 5.5 Anchors 属性值的含义

属性值	含义
akLeft	当 akLeft 值为 true 时,面板的左边随容器的左边框移动而移动,即面板左边与容器的左边框的距离不会随容器大小发生变化而变化。默认值为 true
akTop	当 akTop 值为 true 时,面板的上边随容器的上边框移动而移动,即面板上边与容器的上边框的距离不会随容器大小发生变化而变化。默认值为 true
akRight	当 akRight 值为 true 时,面板的右边随容器的右边框移动而移动,即面板右边与容器的右边框的距离不会随容器大小发生变化而变化。默认值为 false
akBottom	当 akBottom 值为 true 时,面板的下边随容器的下边框移动而移动,即面板下边与容器的下边框的距离不会随容器大小发生变化而变化。默认值为 false

6. Style 属性

在对象观察器展开 Localizable 标签,进一步展开 Font 属性,找到子属性项 Style。该属性项又包含四个子属性,即 fsBold、fsItalic、fsUnderline、fsStrikeOut,它们的作用如下:

- fsBold 属性值为 true,则表示 Label 标签中的文本字体为粗体。
- fsItalic 属性值为 true,则表示字体为斜体。
- fsUnderline 属性值为 true,则表示字体下划线。
- fsStrikeOut 属性值为 true,则表示字体加删除线。


7. Size 属性和 Color 属性

这两项属性是 Font 属性的子属性,而 Font 属性则位于 Localizable 标签页上。它们分别用于设置文本字体的大小和颜色。也可以用代码动态设置,例如:

```
Edit1.font.Color := clRed;      //把编辑框中字体设为红色
Edit1.font.size:=18;           //把编辑框中字体设为 18 号
```

另外,Label 组件的方法几乎在 Button 组件的方法中都可以找到,如 OnClick 等,所以 Label 组件方法的使用可以参照 Button 组件方法的使用说明。但是,Label 组件主要是用作文本显示的“板子”,一般很少使用它的这些方法。

5.3.2 Edit 组件

Edit 组件  也叫做编辑框,它不但可以用于显示文本而且还可以用于编辑文本,它可以为在组件中显示或输入的文本提供单个格式化样式。Edit 组件常用的属性和方法说明如下。

1. Edit 组件的属性

(1) Text 属性。组件显示的文本就是 Text 属性的当前值,最多可以输入 32KB 的字符。在对象观察器中该属性位于 Localizable 标签页上。可用代码动态改变或获取编辑框中的字符,例如:

```
Edit1.Text := '中华人民共和国';    //设置编辑框  
str:= Edit1.Text;                  //获取编辑框中的内容
```

(2) ReadOnly 属性。ReadOnly 属性值为布尔型,它位于 Input 标签页上。当 ReadOnly 属性值为 true 时编辑框为只读(可复制等),不能输入(但可用程序写入),否则 ReadOnly 属性值为 false(默认值)时编辑框为既可读也可写(输入)。

(3) Constraints 属性。Constraints 属性位于 Layout 标签页上,它有四个子属性:

- MaxHeight 用于设置编辑框的最大高度,在设计时其高度(Height)不能超过该值(超过部分将被截取)。默认值为 0,表示高度设计不受此限制。
- MaxWidth 用于设置编辑框的最大宽度,在设计时其宽度(Width)不能超过该值(超过部分将被截取)。默认值为 0,表示宽度设计不受此限制。
- MinHeight 用于设置编辑框的最小高度,在设计时其高度(Height)不能小于该值。默认值为 0,表示高度设计不受此限制。
- MinWidth 用于设置编辑框的最小宽度,在设计时其宽度(Width)不能小于该值。默认值为 0,表示宽度设计不受此限制。

(4) Height 属性和 Width 属性。Height 属性和 Width 属性分别用于设置编辑框的高度和宽度,但其受约束于 Constraints 属性。

(5) Ctl3D 属性。Ctl3D 属性是布尔型,位于 Legacy 标签页上。当 Ctl3D 属性值为 true(默认值)时表示编辑采用 3D 效果,如果为 false 表示编辑采用平面效果。

(6) Style 属性。在对象观察器展开 Localizable 标签,进一步展开 Font 属性,找到子属性 Style。该属性项又包含四个子属性,即 fsBold、fsItalic、fsUnderline、fsStrikeOut,它们的作用如下:

- fsBold 属性值为 true,则表示 Label 标签中的文本字体为粗体。
- fsItalic 属性值为 true,则表示字体为斜体。
- fsUnderline 属性值为 true,则表示字体下划线。
- fsStrikeOut 属性值为 true,则表示字体加删除线。

这四个子属性值的设置也可以在运行时用代码动态设置。例如,要把文本框中的字体设为粗体、下划线和删除线,就可用下列语句:

```
Edit1.Font.Style := [fsUnderline,fsStrikeOut,fsBold];  
//如果还添加 fsItalic,则字体还可以是斜体(跟添加顺序无关)
```

(7) BorderStyle 属性。BorderStyle 属性位于 Visual 标签页上,其值为 bsNone 时,编辑框没有边框;为 bsSingle(默认值)时则编辑框有边框。

(8) Size 属性和 Color 属性。这两项属性是 Font 属性的子属性,而 Font 属性则位于 Localizable 标签页上(在 Visual 标签页上也有)。它们分别用于设置文本字体的大小和颜色。也可以用代码动态设置,例如:

```
Edit1.font.Color := clGreen;    //把编辑框中字体设为绿色
```

```
Edit1.font.size:=20;           //把编辑框中字体设为 20 号
```

(9) PasswordChar 属性。有时编辑框被用作密码输入框，而密码输入一般是保密的。为此，可以把 PasswordChar 属性值设置为“*”，这样在用该编辑框输入字符时它显示的都是“*”（显示星号），这样别人就不知道所输入的内容。当然也可以把 PasswordChar 属性值设置为其他字符，那么在输入时就显示相应的字符。

2. Edit 组件的方法

Edit 组件的所有方法如图 5.9 所示。从图中可以看出，大部分方法都是许多组件共有的，以下主要说明 Edit 组件特有的常用方法。

- OnChange 方法。当 Edit 组件中的文本被改动时该方法被触发。
- OnClick 方法。单击 Edit 组件时该方法被触发。
- OnDblClick 方法。双击 Edit 组件时该方法被触发。
- OnEnter 方法。Edit 组件获得焦点时该方法被触发。
- OnExit 方法。与 OnEnter 方法相反，当 Edit 组件失去焦点时该方法被触发。
- OnContextPopup 方法。对 Edit 组件右击时该方法被触发。
- OnKeyPress 方法。当 Edit 组件获得焦点并且有键被按下时，该方法被触发。

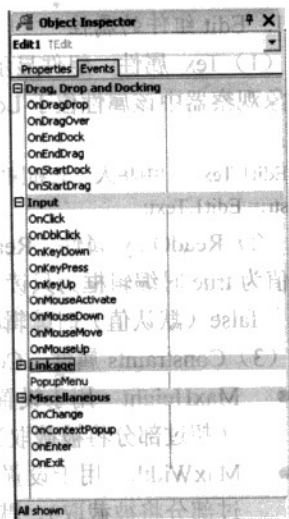



图 5.9 Edit 组件的所有方法

5.3.3 Memo 组件与实例

Memo 组件用于显示和编辑多行文本，其属性和方法很多都是与 Edit 组件的相同。下面主要介绍 Memo 组件特有的一些属性。

1. Lines 属性

Lines 属性主要用于从 Memo 组件读取文本或者向 Memo 组件写入文本，它是一个 TString 类的对象。在程序设计时，如果要在 Memo 组件中添加文本，则可单击该属性右边的省略号按钮 ，将打开 String List Editor 对话框，在此用户可以输入想要在 Memo 窗口显示的文本。

也可以在程序运行时动态添加或删除 Memo 窗口中的文本，这要用到 TString 类的 Add、Delete 和 Insert 等方法。

Lines 属性位于 Localizable 标签页上。

2. WordWrap 属性

WordWrap 属性是布尔型，位于 Miscellaneous 标签页上。当 WordWrap 属性值为 true 时，如果一行上的文本超出了组件的宽度，则文本会自动换行，否则 WordWrap 属性值为 false（默认值）时文本是不会换行的。

3. ScrollBars 属性

ScrollBars 属性用于设置 Memo 组件的滚动条，位于 Miscellaneous 标签页上。它有四个值：ssNone、ssBoth、ssHorizontal、ssVertical。ScrollBars 属性值及其作用如表 5.6 所示。

Memo 组件的方法与 Edit 组件的方法是一样的。以下通过一个例子来熟悉 Memo 组件的使用方法。

表 5.6 ScrollBars 属性值及其作用

属性值	作用
ssNone	垂直和水平方向上都没有滚动条
ssBoth	垂直和水平方向上都有滚动条
ssHorizontal	水平方向上都有滚动条
ssVertical	垂直方向上都有滚动条

例 5.3 Memo 组件的运用范例。

(1) 创建一个新的工程, 命名为 Ex5_3, 窗体单元文件名采用默认值 Unit1.pas。然后设计窗体, 结果如图 5.10 所示, 其中组件的设置情况如表 5.7 所示。

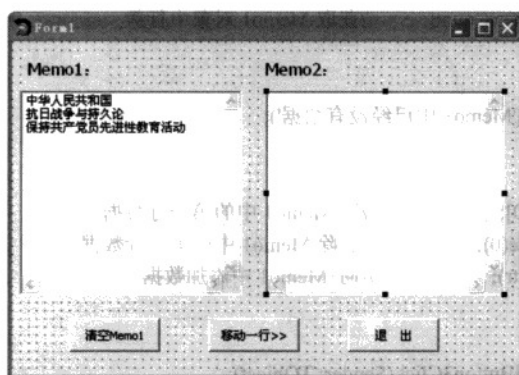


图 5.10 程序 TestMemo 的设计界面

表 5.7 窗体及组件属性设置列表

组件类型	组件名称	属性设置项目	设置结果
Label	Label1	Caption	Memo1:
		Size	13
	Label2	Caption	Memo2:
		Size	13
Memo	Memo 1	Lines	中华人民共和国 抗日战争与持久论 保持共产党员先进性教育活动
		ScrollBars	ssBoth
	Memo 2	Lines	<空>
		ScrollBars	ssBoth
SpeedButton	SpeedButton1	Caption	清空 Memo1
	SpeedButton2	Caption	移动一行>>
	SpeedButton3	Caption	退 出
Form	Form1	Caption	Form1

(2) 为事件处理程序添加代码。这里指的是三个按钮对应的事件处理程序，相应的代码如下：

```
procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
    Memo1.Lines.Clear;
    Memo2.Lines.Clear;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
var Count, i: integer;
var str: string;
begin
    Count := memo1.Lines.Count;    //获取 Memo1 对象中行数
    if Count = 0 then
    begin
        ShowMessage('Memo1 中已经没有数据');
        exit;
    end;
    str := Memo1.Lines[0];        //取 Memo1 中的第一行数据
    Memo1.Lines.Delete(0);        //删除 Memo1 中的第一行数据
    Memo2.Lines.add(str);         //向 Memo1 中添加数据
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
    form1.Close;
end;
```

(3) 编译、运行程序。当程序运行时，每按一次“移动一行>>”按钮，Memo1 中的第一条数据被移动到 Memo2 中，直到 Memo1 中没有数据为止，如图 5.11 所示。

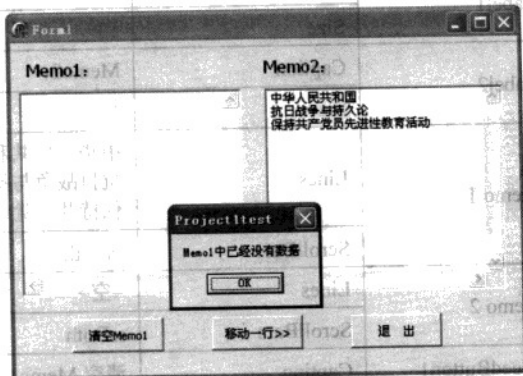


图 5.11 工程 Ex5_3 的运行结果（移动数据）

当然，这个程序的功能较简单，完全可以使用下面两条语句来完成：


```
memo2.Text := memo1.Text;
```




```
Memo1.Lines.Clear;
```

上例的目的是为了了解 Memo 组件，并掌握如何在编程中使用 Memo 组件的一般方法。

5.3.4 MaskEdit 组件

MaskEdit 组件  也是用来显示文本和输入文本的，位于 Localizable 标签页上。它与 Edit 组件基本类似，不同之处在于它有 EditMask 这个特殊的属性，通过这个属性使得 MaskEdit 组件可以显示和编辑具有指定格式的文本。

1. EditMask 属性

单击 EditMask 属性右边的省略号按钮 ，将打开 Input Mask Editor 对话框，如图 5.12 所示。

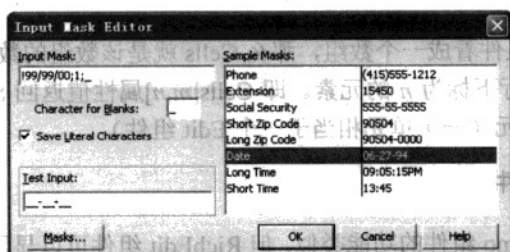


图 5.12 Input Mask Editor 对话框

在该对话框中，Sample Masks 框是预设的格式，当选择其中的一项时，如 Date，则 Input Mask 文本框中就会出现相应的输入格式。如果用户觉得不满意，在此可以编辑格式，直到满意为止。

格式设置完成后，单击 OK 按钮，设定的输入格式将生效，对不符合格式的字符将被过滤等。


2. IsMasked 属性

IsMasked 属性值为布尔型，只读。如果已经设置了输入掩码格式，则 IsMasked 属性为 true；否则为 false。

3. Text 属性和 EditText 属性

两者都是包含 MaskEdit 组件中输入的内容，但它们还是有一点区别：EditText 属性只包含经过格式化后的文字，而 Text 属性则包含实际输入的字符。

5.3.5 SpinEdit 组件

SpinEdit 组件  位于 Samples 标签页上。它由一个上按钮、一个下按钮和一个编辑框共同组成，编辑框用于显示数字，这个数字可以通过上按钮或下按钮根据既定的步长来增加或减少。

1. Text 属性

Text 属性值就是编辑框中的数字，编辑框中数字的变化都将反映到 Text 属性值上。


2. MinValue 属性和 MaxValue 属性

MinValue 属性用于设置编辑框中数字的下界（最小值），而 MaxValue 属性则用于设置编辑框中数字的上界（最大值）。

3. Increment 属性

Increment 属性用于设置编辑框中数字增加或减少的步长。例如, Increment 属性为 5, 也就是说每单击上按钮一次, 编辑框中数字就会增加 5。

5.3.6 StringGrid 组件

StringGrid 组件位于工具面板的 Additional 标签页上, 它是以表格的方式来显示或接收文本输入。该组件常用的属性如下:

1. ColCount 属性和 RowCount 属性


ColCount 属性用于设置或获得 StringGrid 组件的列数, 而 RowCount 属性则用于设置或获得 StringGrid 组件的行数。这两个属性都位于 Miscellaneous 标签页上。

2. Cells[m,n] 属性

如果把 StringGrid 组件看成一个数组, 那么 Cells 就是该数组的数组名, Cells[m,n] 就是表示数组中列下标为 m , 行下标为 n 的元素。即 Cells[m,n] 属性值返回 StringGrid 组件中列、行下标分别为 m 和 n 的单元 (一个单元相当于一个 Edit 组件)。

5.3.7 RichEdit 组件

RichEdit 组件与 Memo 组件的功能类似, 但 RichEdit 组件可以显示、编辑具有丰富文本格式的文件, 如 RTF 文件, 能够表达的文本样式比 Edit 和 Memo 要复杂得多。它相当于 Windows 中的“写字板”。

RichEdit 组件位于工具面板的 Win32 标签页上, 其主要属性和方法说明如下:

(1) DefAttributes 属性。该属性用于设置默认的字符格式, 包括字体、字符颜色、字号等。

(2) DefaultConverter 属性。该属性用于指定一个默认的转换器, 当 RichEdit 组件遇到一个不能识别的文件扩展名时, 就用转换器转换成 RTF 格式。

(3) HideScrollBars 属性。该属性为布尔型属性, 当其值为 true 时, 如果不需要用到滚动条, 那么滚动条将自动隐藏。

(4) HideSelection 属性。该属性值也为布尔型。当取值为 true 时, 如果 RichEdit 组件失去焦点, 则该组件对应的编辑器中被选中的文本不再保持被选中状态 (只是视觉上没有被选中——没有变蓝色, 而实际上组件还是默认被选中); 而当该属性取值为 false 时, 如果 RichEdit 组件失去焦点, 则该组件对应的编辑器中被选中的文本仍然保持被选中状态。

(5) Lines 属性。Lines 属性是一个 TString 对象, 通过它可以设置和修改编辑器中的每一个字符。

(6) Paragraph 属性。该属性用于设置文本的段落格式, 包括对齐、缩进、编号、制表位等。

(7) PlainText 属性。该属性值为布尔型, 用于设置读取或写出文本的格式。如果取值为 true, 则表示将以纯文本格式进行读入或写出; 如果取值为 false, 则表示将以 RTF 格式进行读入或写出。

(8) SelAttributes 属性。该属性用于设置组件中被选中文本的字符格式。

(9) SelStart 属性。该属性返回当前被选中的文本中第一个字符所在的位置, 如果当前没

有文本被选中, 则该属性值为光标所在字符的位置。

也可以对该属性进行设置, 这须与 SelLength 属性搭配使用。

(10) SelLength 属性。该属性返回被选中的字符数。如果对该属性进行设置, 如把它设置为 n , 则从 SelStart 属性值指定的字符位置起选择 n 个字符, 即使这 n 个字符编程被选中状态。

(11) SelText 属性。SelText 属性返回当前被选中的文本。如果对该属性进行写操作, 则写入的字符将替换被选中的字符; 如果当前没有字符被选中, 则写入的字符将插入到光标所在的位置。

(12) OnChange 事件。编辑器中文本被改动时, 该事件被触发。

(13) OnSelectionChange 事件。当文本被选择的状态发生改变时, 该事件被触发。实际上, 只要光标在编辑器中移动位置就会触发该事件。

(14) Clear 方法。清空编辑器中全部的内容。


(15) ClearSelection 方法。清空编辑器中被选中的内容。

(16) FindText 方法。该方法用于在编辑器中查找一个字符串, 如果找到则返回该串第一个字符在编辑器中的位置, 否则返回-1。

(17) Print 方法。该方法用于打印编辑器中所有的文本。

5.4 列表组件

5.4.1 ListBox 组件与实例

ListBox 组件  通常叫做列表框, 位于工具面板上的 Standard 标签中。它可以显示一列或者多列的字符串项, 以供用户选择 (一项或多项)。常用的属性如下:

(1) Columns 属性。Columns 属性位于 Miscellaneous 标签页上, 其值用于设置列表框中要显示文本的列数。默认值为 0, 表示列表框以单列显示。

(2) BorderStyle 属性。BorderStyle 属性位于 Visual 标签页上, 它有两个值, 即 bsNone 和 bsSingle。当其值为 bsNone 时, 编辑框没有边框; 为 bsSingle (默认值) 时则编辑框有边框。

(3) ItemHeight 属性。用于设置列表框中各项的高度, 相当于 Word 中行距的调整。其有效前提是, Style 属性值设为 lbOwnerDrawFixed。该属性位于 Miscellaneous 标签页上。其动态设置格式为:

```
ListBox1.ItemHeight := n; //n 为整数
```

(4) Sorted 属性。Sorted 属性用于设置列表框中文本的排序方式, 属性值为布尔型。当 Sorted 属性值为 true 时表示对列表框中的字符串项进行排序。默认值为 false。其代码设置格式为:

```
ListBox1.Sorted := true;
```

该属性位于 Miscellaneous 标签页上。

(5) MultiSelect 属性。MultiSelect 属性值为布尔型, 当其值为 true 时可同时选择列表框中多项, 为 false (默认值) 时一次只能选择一项。它位于 Miscellaneous 标签页上。其代码设置格式为:

```
ListBox1.MultiSelect := true;
```

(6) **SelCount** 属性。**SelCount** 属性为只读属性, 当 **MultiSelect** 属性值为 **false** (不允许选择多项) 时, 则该属性总是返回 -1; 当 **MultiSelect** 属性值为 **true** 时, 该属性返回被选中项目的个数, 不选时则返回 0。其引用格式为:


`n := ListBox1.SelCount; //n 为整型`

(7) **Selected** 属性。该属性为布尔型只读属性, 当它的值为 **true** 时表示列表框中已有文本项被选择。另外, **Selected** 属性还有一种形式: **Selected[i]**, 如果它的值为 **true** 则表示索引号为 *i* 的项已被选中。其引用格式为:

`b := ListBox1.Selected[i]; //b 为布尔类型`

(8) **ItemIndex** 属性。该属性为只读, 返回列表框中被选中的项目的索引号。如果有多项被选择, 则返回获得焦点的那一项的索引号。其引用格式为:

`n := ListBox1.ItemIndex; //n 为整型`

(9) **Items** 属性。**Items** 属性位于 **Localizable** 标签页上, 用来设置列表框中的内容, 方法是: 单击 **Items** 属性右边的省略号按钮 , 打开如图 5.13 所示的对话框。

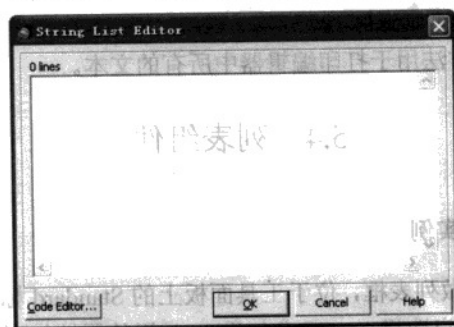


图 5.13 String List Editor 对话框

此对话框主要用于在程序设计阶段对列表框进行初始值设置。下面介绍在程序运行过程中对列表框内容动态改变的方法, 它主要是采用 **TString** 类提供的方法来完成。

一个 **Items** 项就是一个 **TString** 类对象, 下面举例说明该对象方法的使用方式:

- `n := ListBox1.Items.Count;` //返回列表框中项的总数
- `Listbox1.Items.Clear;` //清除列表框中所有的项, 等价于 `Listbox1.Clear;`
- `Listbox1.Items.Delete(i);` //删除列表框中的第 *i* 行
- `Listbox1.Items.Insert(i,'str1');`
//把字符串 `'str1'` 插入到索引号为 *i* 的位置中, *i* 及 *i* 以后的项往后移
- `Listbox1.Items.Add('str2');` //在列表框中添加字符串 `str2`
- `Listbox1.Items.Exchange(i,j);` //交换索引号为 *i* 和 *j* 的项
- `str:= ListBox1.Items.Text;` //返回列表框中的内容
- `str := ListBox1.Items[i];` //返回列表框中索引号为 *i* 的项

其中, *n* 为整型, *str* 为字符串型。

以下通过一个例子来熟悉 **ListBox** 组件的使用方法。

例 5.4 在一个程序界面中添加两个列表框, 即 **ListBox1** 和 **ListBox2**, 程序实现下述功能:

- 为 **ListBox1** 添加项。

- 选择 ListBox1 中的项，然后把被选中的项移到 ListBox2 中。
- 双击 ListBox1 中的项，可以把该项移到 ListBox2 中。
- 可以动态调整 ListBox1 和 ListBox2 中项的行距。
- 清空两个 ListBox 框中的项。

按照下列步骤创建该程序：

- (1) 创建一个新的工程，命名为 Ex5_4.dpr，窗体单元文件取默认名称 Unit1.pas。
- (2) 设计窗体，结果如图 5.14 所示，其中组件的设置情况如表 5.8 所示。



图 5.14 程序 Ex5_4.dpr 的设计界面

表 5.8 窗体及组件属性设置列表

组件类型	组件名称	属性设置项目	设置结果
Label	Label1	Caption	ListBox1:
		Size	13
	Label2	Caption	ListBox2:
		Size	13
	Label3	Caption	输入要添加的项:
		Size	10
Edit	Edit1	Text	(空值)
ListBox	ListBox1	Style	lbOwnerDrawFixed
		ItemHeight	12
		MultiSelect	True
		Items	白日依山尽, 黄河入海流。 欲穷千里目, 更上一层楼。

续表

组件类型	组件名称	属性设置项目	设置结果
	ListBox2	Style	lbOwnerDrawFixed
		ItemHeight	12
		MultiSelect	True
		Items	(空值)
Button	Button1	Caption	移动被选中的项
	Button2	Caption	清除两个表中的项
	Button3	Caption	在列表框 1 中添加项
SpinEdit	SpinEdit1	MinValue	12
		MaxValue	40
Form	Form1	Caption	使用 ListBox 组件

(3) 编写单击事件和双击事件的处理过程，最后得到窗体单元文件代码如下。

```
unit Unit1;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, StdCtrls, Mask, Buttons, Grids, Calendar, Spin;
```

```
type
```

```
TForm1 = class(TForm)
```

```
  Edit1: TEdit;
```

```
  Button1: TButton;
```

```
  ListBox2: TListBox;
```

```
  Label1: TLabel;
```

```
  Label2: TLabel;
```

```
  Label3: TLabel;
```

```
  Button2: TButton;
```

```
  Button3: TButton;
```

```
  SpinEdit1: TSpinEdit;
```

```
  ListBox1: TListBox;
```

```
  procedure SpinEdit1Change(Sender: TObject);
```

```
  procedure Button3Click(Sender: TObject);
```

```
  procedure Button2Click(Sender: TObject);
```

```
  procedure ListBox1DbClick(Sender: TObject);
```

```
  procedure Button1Click(Sender: TObject);
```

```
private
```

```
  { Private declarations }
```

```
public
```

```
  { Public declarations }
```

```
end;
```

```
var
    Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject); //移动被选中的项
var
    str:string;
    Count,i:integer;
begin
    Count := ListBox1.Items.Count;
    i := 0;
    while i<= Count-1 do
    begin
        if ListBox1.Selected[i] then
        begin
            str := ListBox1.Items[i];
            ListBox2.Items.Add(str);
            ListBox1.Items.Delete(i);
            Count:=Count-1;
        end
        else
            i := i+1;
        end;
    end;
end;

procedure TForm1.ListBox1DbClick(Sender: TObject); //双击移动项
var
    i:integer;
    str:string;
begin
    i:=ListBox1.ItemIndex;
    str := ListBox1.Items[i];
    ListBox2.Items.Add(str);
    ListBox1.Items.Delete(i);
end;

procedure TForm1.Button2Click(Sender: TObject); //清除两个 ListBox 中的项
begin
    ListBox1.Clear;
    ListBox2.Items.Clear;
end;

procedure TForm1.Button3Click(Sender: TObject); //为 ListBox1 中添加项
begin
    ListBox1.Items.Add(Edit1.text);
```

```

Edit1.Text := "";
end;

procedure TForm1.SpinEdit1Change(Sender: TObject); //调整 ListBox 中项的行距
begin
    ListBox1.ItemHeight := StrToInt(SpinEdit1.Text);
    ListBox2.ItemHeight := StrToInt(SpinEdit1.Text);
end;
end.

```

(4) 编译、运行程序 Ex5_4.dpr。在该运行程序中，首先在 ListBox1 列表框加入“日照香炉生紫烟，”，然后利用“移动被选中的项”按钮把 ListBox1 列表框中的两项移到 ListBox2 列表框中，最后通过 SpinEdit 组件调整两个列表框中各项之间的距离，结果如图 5.15 所示。

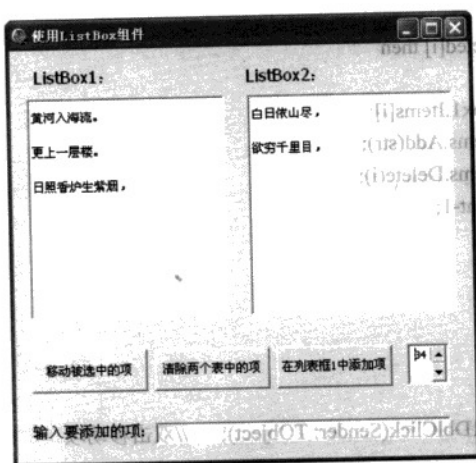



图 5.15 程序 Ex5_4.dpr 运行界面

5.4.2 ComboBox 组件

ComboBox 组件  通常叫做组合框，位于工具面板中的 Standard 标签页上。它与 ListBox 组件的功能基本类似，只是显示方式不同：ListBox 组件是一个“窗口”，而 ComboBox 组件则由一个编辑框和一个下拉列表框组成。与 ListBox 组件不同的是，在 ComboBox 组件中只能选择一个项。

ComboBox 组件的常用属性如下：

(1) Text 属性。Text 属性用于设置或返回组合框中显示的文本，但仅在 csSimple 和 csDropDown 风格的组合框有效。它位于 Localizable 标签中。代码访问格式为：

```
str := ComboBox1.Text; // 获取组合框中的内容
```

```
ComboBox1.Text := str; // 设置组合框中的内容，其中 str 为字符串变量
```

例如，可以在 ComboBox 组件的 OnChange 方法中添加以下语句：

```
Listbox1.Items.Add(ComboBox1.Text);
```

这样，使得每选一次都会把被选中的项添加到 ListBox1 中。

(2) DropDownCount 属性。DropDownCount 属性用于设置在不用滚动条时组合框能够显

示的最大项数，默认值为 8。该属性位于 Miscellaneous 标签页上。

(3) ItemIndex 属性。ItemIndex 属性用于设置组合框显示的项 (Text 属性的值) 的索引号。如果该属性的值被设置为 n，那么就显示索引号为 n 的项 (被设置为 Text 属性的值)。另外，在程序运行时可以通过读取该属性的值来获知被显示的项的索引号，其引用格式为：

```
n := ComboBox1.ItemIndex; //n 为整型
```

ItemIndex 属性位于 Miscellaneous 标签页上。

(4) CharCase 属性。CharCase 属性用于设置组合框显示时文本的大小写，它有三个值，即 ecNormal、ecLowerCase 和 ecUpperCase。当 CharCase 属性值为 ecNormal 时表示用字符原有的大小写格式显示；当为 ecLowerCase 时表示所有的字符用小写显示；当为 ecUpperCase 时表示所有的字符用大写显示。


(5) MaxLength 属性。用于设置组合框中可输入的最多字符数，默认值为 0，表示没有限制。该属性位于 Miscellaneous 标签页上。

(6) Style 属性。Style 属性用于设置组合框的显示风格，它位于 Miscellaneous 标签页上，有五个值，表 5.9 对其含义和作用作了说明。


表 5.9 Style 属性值及其含义

属性值	含义
csDropDown	组合框的编辑框可写，且下拉列表框中的每一项都是高度相等的字符串
csDropDownList	组合框的编辑框只读，编辑框中的内容只能从下拉列表框中选择
csOwnerDrawFixed	组合框的编辑框只读，下拉列表框中项的高度由属性 ItemHeight 确定
csOwnerDrawVariable	组合框的编辑框只读，下拉列表框中项的高度可以不一致
csSimple	编辑框可读可写，下拉列表框变成一般的列表框了 (不下拉)，但可从中选择项目

5.4.3 TreeView 组件

TreeView 组件  位于工具面板中 Win32 标签页上。它在表示树型的数据结构时非常有效，TreeView 组件典型的应用例子就是 Windows 操作系统的资源管理器。

下面介绍 TreeView 组件的一些主要应用。

单击 Items 属性右边的省略号按钮 ，将弹出 TreeView Items Editor 对话框，在此对话框中单击 New Item 按钮后其右边的“Text”文本框会变成可写状态，它用于设置树的结点标识符，在此输入“1”。

单击 New SubItem 按钮，在 Text 文本框中输入 1.1，这样就创建了结点“1”子结点“1.1”；接着，单击 New Item 按钮 (而不是“New SubItem”按钮，这时要保证结点“1.1”呈被选中状态)，在 Text 文本框中输入“1.2”，这样又把结点“1.1”的兄弟结点“1.2”创建完毕。

用上述介绍的创建子结点和兄弟结点的方法进一步扩展该树，结果如图 5.16 所示。

对于上面创建的树，程序员也可以用代码来动态实现，实现的代码如下：

```
TreeNode1:=TreeView1.Items.Add(nil,'1');           //创建根节点“1”
TreeView1.Items.AddChild(TreeNode1,'1.1');         //创建节点“1”的子节点“1.1”
TreeNode2:=TreeView1.Items.AddChild(TreeNode1,'1.2'); //创建节点“1”的子节点“1.2”
TreeNode2:=TreeView1.Items.AddChild(TreeNode2,'1.2.1'); //创建节点“1.2”的子节点“1.2.1”
```



```
TreeView1.Items.Add(TreeNode2,'1.2.2');  
TreeNode1:=TreeView1.Items.Add(nil,'2');
```

//创建节点“1.2.1”的兄弟节点“1.2.2”
//创建节点“1”的兄弟节点“2”

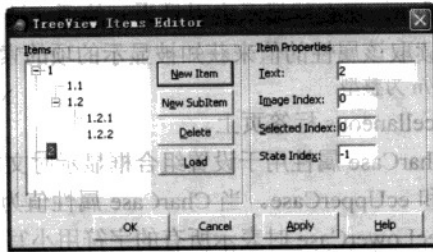



图 5.16 TreeView Items Editor 对话框

5.4.4 ListView 组件

ListView 组件位于工具面板中 Win32 标签页上。它可按照大图标、小图标、列表和详细资料 4 种方式显示。Windows 操作系统的资源管理器左边的方框就是 ListView 组件典型的应用。

(1) Columns 属性。用于设置在 ListView 组件中要显示的列数，该属性位于 Miscellaneous 标签页上。

(2) ViewStyle 属性。用于设定 ListView 组件的显示风格，即确定大图标、小图标、列表或详细资料四种方式之一。它位于 Miscellaneous 标签页上。当 ViewStyle 属性取值为 vsIcon、vsList、vsReport 或 vsSmallIcon，ListView 组件将分别采用大图标、列表、详细资料、小图标显示其中的信息。

(3) Items 属性。ListView 组件的 Items 属性与 TreeView 组件类似，单击该属性右边的省略号按钮，将打开 ListView Items Editor 对话框，如图 5.17 所示。

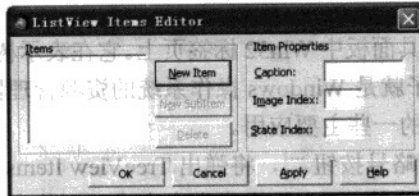


图 5.17 ListView Items Editor 对话框


对该对话框的操作与对 TreeView Items Editor 对话框的操作基本类似，其中，单击 New Item 按钮将在 ListView 组件中创建每一个记录的第一列，然后单击 New SubItem 按钮将创建第二列、第三列等。

5.5 对话框组件

很多 Delphi 组件都是利用 Windows 公共控件进行开发而得到的，实际上是 Windows 控件的 Delphi 包装。Delphi 提供的对话框组件大多都属于此类，本节将介绍几个常用的对话框。

5.5.1 OpenFileDialog 组件

OpenDialog 组件实际上是对 Windows 打开文件对话框的包装,利用它可以轻易地实现文件的浏览和打开。通常称为打开文件对话框。该组件位于工具面板中 Dialogs 标签上,其常用属性和方法包括以下几种:

(1) Filter 属性。Filter 属性位于 Database 标签下,用于将那些不需要的文件类型过滤掉。设置的方法是:单击 Filter 属性的省略按钮 ,弹出 Filter Editor 对话框。此对话框分为两栏:左边用于设置说明性文字,右边才是过滤文件的条件。如果将过滤条件设置为如图 5.18 所示,那么运用这个对话框只能选择.txt 文件(文本文件)、.rtf 文件和.doc 文件(Word 文件)。此属性可以用代码设置,例如:

```
OpenDialog1.Filter := 'Word files(*.doc)|*.doc';
```

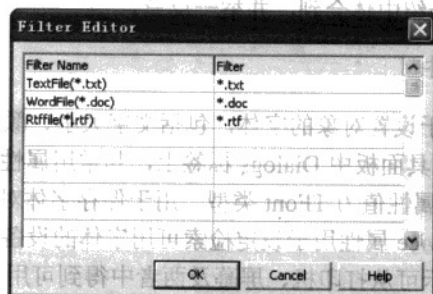


图 5.18 Filter Editor 对话框

说明性文字和过滤文件的条件之间要用竖线“|”隔开。

(2) File 属性。File 属性值是 TString 类型,它返回所有被选中的文件名字符串及其完整的路径名(如要选择多个,则必须把 Options 属性值设置为包含 ofAllowMultiSelect)。

(3) FileName 属性。该属性用于设置“打开”对话框中“文件名”文本框中的内容。当在“打开”对话框中选定某一个文件并单击“打开”按钮后,FileName 属性将返回该文件的完整路径(包括文件名);如果有多个文件被选中,则 FileName 属性将返回获得焦点的那个文件路径;如果选中的文件都没有获得焦点,则 FileName 属性将返回第一个被选中的文件路径。

(4) Title 属性。该属性用于设置“打开”对话框的标题,可以用代码动态设置,例如:
`OpenDialog1.Title := '打开 word 文件';`


(5) FilterIndex 属性。当过滤条件设为多个的时候,FilterIndex 属性用于设置在“文件类型”下拉列表框中默认显示的文件类型,其值是整型。例如,如果按照图 5.18 所示的设置条件,并且把 FilterIndex 属性值设为 3,那么“文件类型”对应的下拉列表框中默认显示的是“RtfFile(*.rtf)”。

(6) InitiaDir 属性。InitiaDir 属性用于设置“打开”对话框打开的默认目录,如果不设置(空值),那么将默认打开当前工作的目录。

(7) Execute 方法。Execute 方法被触发时打开 OpenFileDialog 对话框,其语法格式为:

```
OpenDialog1.Execute();
```


5.5.2 SaveDialog 组件

SaveDialog 组件  是在 Windows 保存对话框的基础上开发而得到的，用于设置文件名并保存文件。该组件位于工具面板中 Dialogs 标签上。通常称为保存文件对话框。

该组件的属性和方法与 OpenFileDialog 组件的几乎相同，不同的是，Options 属性值中多一个可选值——OfOverwritePrompt。如果 Options 属性值选中了 OfOverwritePrompt (OfOverwritePrompt 值设为 true)，那么在保存文件时，如果遇到要保存的文件与已有文件重名时它将给出是否要覆盖已存在文件的提示。这样，可以有效地避免误操作带来的损失。

此外，Dialogs 标签上除了 OpenFileDialog 组件和 SaveDialog 组件，类似的组件还包括 OpenPictureDialog、SavePictureDialog、OpenTextFileDialog、SaveTextFileDialog，它们的属性和方法与 OpenFileDialog 组件和 SaveDialog 组件的属性和方法非常相似，读者可以从 OpenFileDialog 组件和 SaveDialog 组件的介绍中体会到，并举一反三。

5.5.3 FontDialog 组件

FontDialog 组件  用于设置对象的字体，包括文字大小、颜色、样式等属性，通常称为字体对话框。该组件位于工具面板中 Dialogs 标签上，其常用属性和方法包括以下几种：

(1) Font 属性。Font 属性值为 TFont 类型，用于保存字体对话框的设置结果。


(2) Device 属性。Device 属性用于设定检索可用字体的设备，它包含三个值：fdPrinter、fdScreen、fdBoth，分别表示可从打印机、屏幕及两者中得到可用字体。

(3) MinFontSize 属性和 MaxFontSize 属性。这两个属性分别用来设置所允许的最大和最小字体尺寸，0 表示没有此限制。

(4) Execute 方法。Execute 方法被触发时打开 FontDialog 对话框，其语法格式为：

```
FontDialog1.Execute();
```



5.5.4 ColorDialog 组件

ColorDialog 组件  用于设置对象的颜色属性，包括自定义颜色值。其常用属性包括以下两种：


(1) Color 属性。Color 属性对 ColorDialog 组件的作用与 Font 属性对 FontDialog 组件的作用类似，它用于设置对话框的初始颜色，更重要的是它用于保存对话框设置后的结果，值为 TColor 类型。


(2) CustomColors 属性。CustomColors 属性值是 TString 类型，用于初始化或保存用户自定义的颜色。

5.5.5 PrintDialog 组件和 PrintSetupDialog 组件

PrintDialog 组件  用于打开 Windows 打印选项对话框，向打印机传送任务；PrintSetupDialog 组件  则用于打开打印设置对话框，设置打印任务。这两个组件设置的属性不多，一般只要调用它们的 execute 方法即可打开相应的对话框。对话框的设置和运用与 Word 中的打印对话框类似。

5.5.6 FindDialog 组件和 ReplaceDialog 组件

FindDialog 组件  是一种提供文本查找功能的组件，其 FindText 属性用于设置待查找的文本。

ReplaceDialog 组件  除了具有 FindDialog 组件的查找功能外，还可以用于替换字符串。

5.5.7 开发实例——对话框的应用

本节以制作一个文本编辑器为例，主要介绍常用对话框的使用方法。

该示例程序的基本功能是：

- 可以打开和保存文本文件。
- 能够设置文本的字体。
- 可以设置编辑器的背景颜色。

为此，创建一个工程，命名为 Ex5_5.dpr，其窗体单元文件名依然取默认值 Unit1.pas。然后在窗体上添加一个 Memo 组件、五个 Button 组件，以及 OpenFileDialog、SaveDialog、FontDialog、ColorDialog 组件各一个，并对它们的属性进行设置，结果如表 5.10 所示。

表 5.10 窗体及组件属性设置

组件类型	组件名称	属性设置项目	设置结果
Memo	Memo1	Lines	Memo1
OpenDialog	OpenDialog1	Filter	Word files (*.doc) *.doc Textfile (*.txt) *.txt
		FilterIndex	2
		Options	[ofHideReadOnly, ofEnableSizing]
SaveDialog	SaveDialog1	Filter	(空值，用代码设置)
		FilterIndex	(空值，用代码设置)
		Options	[ofOverwritePrompt, ofHideReadOnly, ofEnableSizing]
FontDialog	FontDialog1		(所有属性采用默认值)
ColorDialog	ColorDialog1		(所有属性采用默认值)
Button	Button1	Caption	打开文件
	Button2	Caption	保存文件
	Button3	Caption	字体设置
	Button4	Caption	颜色设置
	Button5	Caption	退 出
Form	Form1	Caption	对话框的应用示例

属性设置完毕以后，进一步设计程序界面（窗体设计），结果如图 5.19 所示。

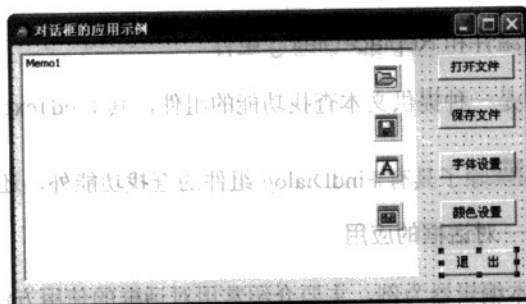


图 5.19 Ex5_5.dpr 程序界面设计效果

最后编写各个按钮事件的处理程序，得到 Ex5_5.dpr 程序的单元文件代码如下：

```
unit Unit1;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
```

```
Dialogs, ComCtrls, StdCtrls, ImgList;
```

```
type
```

```
TForm1 = class(TForm)
```

```
Button1: TButton;
```

```
Memo1: TMemo;
```

```
OpenDialog1: TOpenDialog;
```

```
SaveDialog1: TSaveDialog;
```

```
FontDialog1: TFontDialog;
```

```
ColorDialog1: TColorDialog;
```

```
Button2: TButton;
```

```
Button3: TButton;
```

```
Button4: TButton;
```

```
Button5: TButton;
```

```
procedure Button4Click(Sender: TObject);
```

```
procedure Button3Click(Sender: TObject);
```

```
procedure Button2Click(Sender: TObject);
```

```
procedure Button5Click(Sender: TObject);
```

```
procedure Button1Click(Sender: TObject);
```

```
private
```

```
{ Private declarations }
```

```
public
```

```
{ Public declarations }
```

```
end;
```

```
var
```

```
Form1: TForm1;
```

```
nn: integer;
```

```
implementation
```

```
{ $R *.dfm }
```

```
procedure TForm1.Button1Click(Sender: TObject); //打开文件
begin
if OpenFileDialog1.Execute() then
begin
Memo1.Clear;
Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
form1.Close
end;

procedure TForm1.Button2Click(Sender: TObject); //保存文件
begin
SaveDialog1.Filter := 'Word files(*.doc)|*.doc|Txtfile(*.txt)|*.txt';
//用代码设置过滤条件
SaveDialog1.FilterIndex := 2; //设置默认显示的文件类型为*.txt 文件
if SaveDialog1.Execute() then
begin
Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
end;

procedure TForm1.Button3Click(Sender: TObject); //字体设置
begin
if FontDialog1.Execute() then
begin
Memo1.Font:=FontDialog1.Font;
end;
end;

procedure TForm1.Button4Click(Sender: TObject); //背景颜色设置
begin
if ColorDialog1.Execute() then
begin
Memo1.Color := ColorDialog1.Color;
end;
end;
end.
```

Ex5_5.dpr 程序运行界面如图 5.20 所示。在该程序中，可以打开一个已经存在的文本文件，然后对其进行编辑，最后将其保存。在编辑过程中，可以设置文本的字体及设置编辑器的背景颜色等。

该编辑器不能打开 doc 文件（打开后会出现乱码），但为了下面的深入学习，过滤条件允许 doc 文件在对话框中显示。

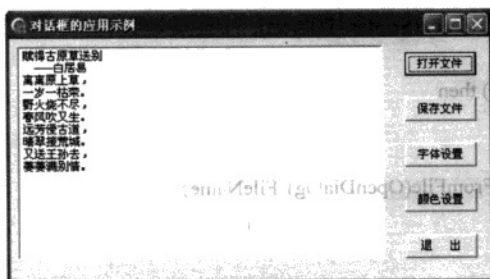




图 5.20 Ex5_5.dpr 程序运行界面

5.6 分类组件

5.6.1 GroupBox 组件和 RadioGroup 组件


GroupBox 组件  通常称为分组框，位于工具面板的 Standard 标签页上。它主要有两方面的作用：一是加强程序界面的外观美和界面设计的逻辑意义，二是组织有内在关联的组件，如 RadioButton 组件放在一个 GroupBox 对象中，它们就自动形成一组，每次只能同时选择一个 RadioButton。注意，GroupBox 组件与放于其中的组件将形成一个整体，而不是独立的。

对于 GroupBox 组件，主要是设置它的 Caption 属性，该属性值也可以为空。

RadioGroup 组件  也位于工具面板的 Standard 标签页上。它的设计外形上与 GroupBox 组件完全相同，但是在设计过程中却有很大的区别。可以把 RadioButton 组件直接拖到 GroupBox 组件中，而 RadioGroup 组件则不行，即使拖了也不在其中——不形成一组。为正确应用 RadioGroup 组件，应了解其下列属性：

(1) Caption 属性。Caption 属性用于设置 RadioGroup 组件的标题，即 Caption 属性值就是 RadioGroup 组件的标题。

(2) Columns 属性。设置 RadioGroup 组件的显示方式，即按几列来显示 RadioButton 组件。

(3) Items 属性。Items 属性位于 Localizable 标签页下，它用于管理 RadioGroup 组件中的 RadioButton 组件，包括创建、修改和删除，方法是：单击 Items 属性右边的省略号按钮 ，将打开 String List Editor 对话框。在此对话框中每添加一行就会产生一个 RadioButton 对象。图 5.21 给出了一个实例。

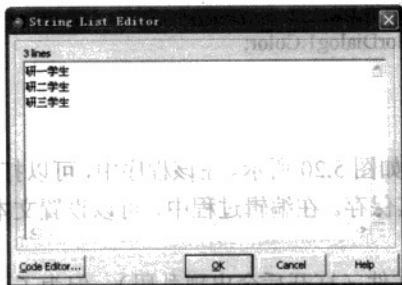


图 5.21 String List Editor 对话框

(4) **ItemsIndex** 属性。**ItemsIndex** 属性位于 **Miscellaneous** 标签页上, 它用于设置 **RadioGroup** 组件中默认选中的 **RadioButton** 对象。其默认值为 -1, 如果其值为 0 则表示组件中默认选中第一个 **RadioButton** 对象; 为 1 则表示默认选中第二个 **RadioButton** 对象, 依此类推。

5.6.2 Panel 组件

Panel 组件^④通常称为面板, 也位于工具面板的 **Standard** 标签页上。它是一种大容器, 虽然具有 **Button** 组件的基本功能, 但很少把它当作 **Button** 组件来使用, 而是利用它来设计不同风格的界面, 加强界面的美观。**Panel** 组件在前面的编程中已经有所涉及, 但主要是运用它的最基本功能。相信充分地挖掘 **Panel** 组件的潜能可以设计出意想不到的界面效果, 本小节进一步介绍 **Panel** 组件的常用属性。

(1) **Caption** 属性。**Caption** 属性用于设置面板上显示的文本, 实际面板上显示的文本就是 **Caption** 属性值。该属性位于 **Action** 标签页中 (在其他的许多标签页中也有)。

(2) **AutoSize** 属性。**AutoSize** 属性位于 **Layout** 标签页中, 其值为布尔型。当 **AutoSize** 属性为 **true** 时, 表示在运行时面板大小将自动根据其中的文字大小来调节; 当 **AutoSize** 属性为 **false** 时, 表示面板大小完全是在设计阶段给定的, 程序运行时不会再次发生改变。

(3) **Align** 属性。**Align** 属性用于设置面板在窗体中的对齐方式, 位于 **Layout** 标签页中。它有七个值: **alBottom**、**alClient**、**alCustom**、**alLeft**、**alNone**、**alRight**、**alTop**, 默认值为 **alNone**。**Align** 属性各值含义见表 5.11。

表 5.11 Align 属性各值的含义

属性值	说明
alBottom	靠窗体的底部对齐, 同时在水平方向上顶到窗体的左右边
alClient	占满整个窗体
alCustom	自定义方式, 由用户设定其大小
alLeft	靠窗体左边对齐, 同时在垂直方向上顶到窗体的上下边
alNone	没有用任何对齐方式
alRight	靠窗体右边对齐, 同时在垂直方向上顶到窗体的上下边
alTop	靠窗体的顶部对齐, 同时在水平方向上顶到窗体的左右边

(4) **Color** 属性。**Color** 属性用于设置面板的颜色, 位于 **Visual** 标签页中。当单击 **Color** 属性右边的下拉列表框时就可以设置相应的颜色, 可供选用的颜色非常丰富。

(5) **Font** 属性。**Font** 属性也位于 **Visual** 标签页中, 包含若干个子属性, 它们用于设置面板上文本字符的大小 (**Size**)、颜色 (**Color**) 及所采用的字符集 (**Charset**) 等。

(6) **Layout** 属性。**Layout** 属性用于设置文本字符在面板中垂直方向上的位置, 它有三个值: **tlCenter**、**tlTop**、**tlBottom**。当其值为 **tlCenter** 时表示文本字符将居于面板的中间 (垂直方向), 当为 **tlTop** 时表示文本字符将居于面板的顶部, 而为 **tlBottom** 时则表示居于面板的底部。它位于 **Miscellaneous** 标签页中。

(7) **Alignment** 属性。**Alignment** 属性用于设置文本字符在面板中水平方向上的位置, 它有三个值: **taCenter**、**taLeftJustify**、**taRightJustify**。当其值取为 **tlCenter** 时表示文本字符将居于

面板的中间(水平方向), 当为 `taLeftJustify` 时表示文本字符将对齐面板的左边, 而为 `taRightJustify` 时则表示对齐面板右边。它位于 `Visual` 标签页中。

5.7 菜单组件

5.7.1 MainMenu 组件

菜单是 GUI 应用程序的一个重要组成部分。在 Delphi 中提供了 `MainMenu` 组件, 用于菜单的设计。`MainMenu` 组件的属性不多, 常用的主要是以下两个属性:

(1) `Items` 属性。`Items` 属性用于设计菜单项, 方法是: 单击该属性右边的省略号按钮 (或者在窗体上双击 `MainMenu` 组件, 或者右击 `MainMenu` 组件并在弹出的快捷菜单中选择 “Menu Designer...” 命令), 将打开菜单设计器 (Menu Designer), 如图 5.22 所示。在菜单设计器中可以创建菜单及其子菜单等, 图 5.22 表示已建立了 `File(F)` 菜单。

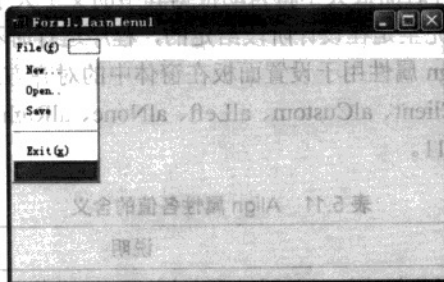


图 5.22 菜单设计器

(2) `Images` 属性。`Images` 属性用于设计主菜单的图标, 要结合 `ImageList` 组件使用。

5.7.2 PopuMenu 组件

`PopuMenu` 组件用于设置弹出式菜单, 弹出式菜单与主菜单的设计方法类似。很多组件都可以设计它的弹出式菜单, 方法是: 把该组件的 `PopuMenu` 属性值设置为已被创建的弹出式菜单对象的名称。

`PopuMenu` 组件常用的属性有以下三个:

(1) `Alignment` 属性。`Alignment` 属性用于设置菜单的弹出位置(相对鼠标), 它有三个值, 即 `paCenter`、`paLeft`、`paRight`。当 `Alignment` 属性取值为 `paCenter` 时, 菜单将以光标点为中心来弹出; 为 `paLeft` 时(默认值), 菜单将在光标点的左边弹出; 为 `paRight` 时在光标点的右边弹出。


(2) `AutoPopu` 属性。`AutoPopu` 属性用于决定弹出式菜单是否自动弹出。当其值为 `true` 时, 右击则弹出快捷菜单, 否则要用程序控制。

(3) `Items` 属性。`Items` 属性与 `MainMenu` 组件中的 `Items` 属性的功能和使用方法相同。

5.8 工具栏和状态栏组件


5.8.1 ToolBar 组件


在标准的 Windows 应用程序中,菜单栏下面就是工具栏。工具栏中的快捷按钮为用户的工作提供了极大的方便,人们已经很习惯了这种风格。在 Delphi 中,可以用 ToolBar 组件来制作工具栏。

ToolBar 组件  位于工具面板中的 Win32 标签页上。设计 ToolBar 组件的方法是:右击 ToolBar 组件,在弹出的快捷菜单中选择 New Button 命令就可以创建一个快捷按钮,如果选择 New Separator 命令,则会创建一个分隔符,一般用于隔开快捷按钮。ToolBar 组件中的按钮大小一致,调节任一个按钮的宽度,其他的按钮也会跟着变动,使得各按钮的大小始终保持一致。

如果想更改 ToolBar 组件在容器中的摆放位置,可以利用 Align 属性来完成。该属性位于 Layout 标签页中,它有七个值:alBottom、alClient、alCustom、alLeft、alNone、alRight、alTop。它们的含义与 Panel 组件中 Align 属性的相同。

5.8.2 StatusBar 组件

StatusBar 组件  通常称为状态栏,位于工具面板中的 Win32 标签页上。StatusBar 组件常用的属性包括以下几个:

(1) Panels 属性。Panels 属性位于 Localizable 标签页中,单击该属性右边的省略号按钮 ,将打开状态栏编辑器,如图 5.23 所示。或者在窗体中右击 StatusBar 组件,在弹出的快捷菜单中选择 Panels Editor...命令。在该编辑器中可以增加、修改和删除状态栏中的面板。

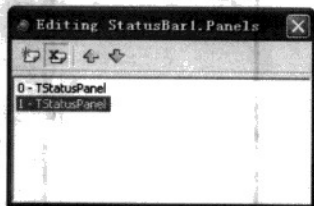


图 5.23 状态栏编辑器

(2) Align 属性。Align 属性位于 Layout 标签页中,用来设置 ToolBar 组件在窗体中的摆放位置。它有七个值:alBottom、alClient、alCustom、alLeft、alNone、alRight、alTop,其含义与 Panel 组件中的 Align 属性相同。

5.9 Timer 组件

Timer 组件也是非常有用的组件,经常用于有间隔地响应和处理系统事件,而避免用手工操作,如显示时间、动态字幕等都是 Timer 组件应用的例子。

Timer 组件的属性很少,在编程中用到的主要是两个属性及惟一的一个方法。

(1) Enable 属性。Enable 属性为布尔型,如果其值为 true (默认值),那么 Timer 组件有效(在工作),否则无效。此属性常用于设置 Timer 组件启动和中止,编码格式为:

```
Timer1.Enabled := true;    //启动  
Timer1.Enabled := false;   //中止
```

(2) Interval 属性。Interval 属性用于设置两个 OnTimer 事件的时间间隔,单位为毫秒(ms),

默认值为 1000ms (1s)。

(3) OnTimer 方法。OnTimer 方法是 Timer 组件唯一的一个方法, Timer 组件每隔一定的时间间隔 (Interval 属性设定) 将调用该方法。程序员可以在该方法中编写相应的处理事件的代码, 以间断性地完成某一些工作。

5.10 创建 GUI 应用程序的窗体及实例开发

下面分别介绍在 Delphi 中创建单文档窗体和多文档窗体这两种窗体的基本方法和步骤。

5.10.1 单文档窗体的创建

在本书前部分中创建的 VCL 应用程序都是单文档窗体类型的。创建这种窗体的步骤如下:

(1) 在 Delphi 2005 的 IDE 中, 选择菜单命令 File → New → Other..., 则会弹出 New Items 对话框。

(2) 在 New Items 对话框中的 Item Categories: 框中选择 Delphi Projects 选项, 在右边的框中则选择 VCL Forms Application 图标, 然后单击 OK 按钮则出现如图 5.24 所示的窗体。

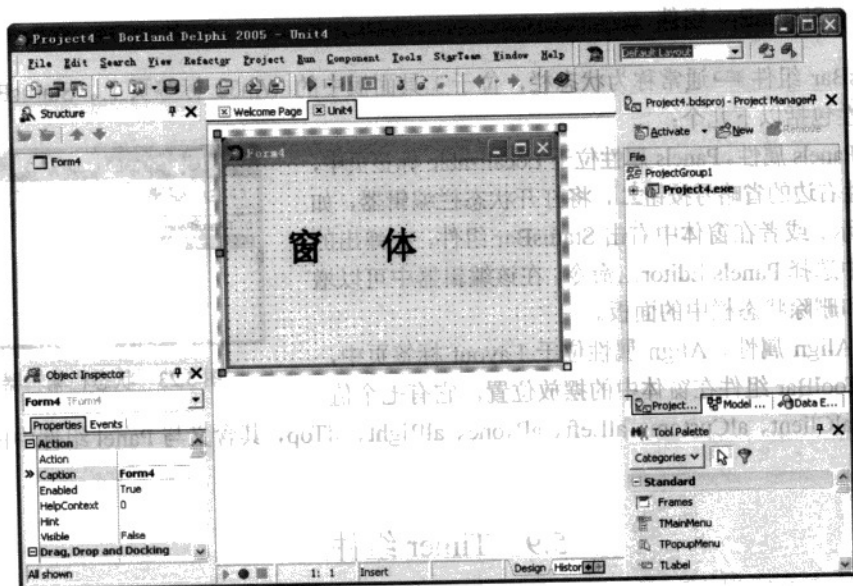



图 5.24 窗体设计界面

(3) 单击工具栏上的  (Save all) 按钮, 保存工程文件, 这包括保存 .pas 文件和 .bdsproj 文件。保存完成后, 会自动产生窗体文件, 即 .dfm 文件。该文件与 .pas 文件类型同名, 其代码如下:

```
object Form1: TForm1
```

```
Left = 0;
```

```
Top = 0;
```

```
Width = 424;
```

```
Height = 304;  
Caption = 'Form1';  
Color = clBtnFace;  
Font.Charset = DEFAULT_CHARSET;  
Font.Color = clWindowText;  
Font.Height = -11;  
Font.Name = 'Tahoma';  
Font.Style = [];  
OldCreateOrder = false;  
PixelsPerInch = 96;  
TextHeight = 13;  
End
```

显然, 这些代码是对窗体属性的描述。一般来说, 不直接修改这些代码(特别是对初学者来说)。随着往窗体上添加元素的增多, 窗体代码也就越来越多。

窗体创建完毕后, 就可以对其进行设计了。这在 2.2.4 节中已经有所介绍, 本章将结合菜单的设计进一步介绍窗体的设计和开发方法。

5.10.2 多文档窗体的创建

多文档窗体出现在多文档应用程序(MDI 应用程序)中, VC 是典型的 MDI 应用程序开发工具。在 Delphi 中, 也可以开发 MDI 应用程序, 在 MDI 应用程序中出现的窗体就是多文档窗体。在多文档窗体的结构中, 有一个窗体称为父窗体, 也叫主窗体; 在父窗体中又可以建立一些窗体, 称为子窗体。显然, 子窗体有多个, 它们之间地位平等, 彼此没有包含关系, 但所有子窗体都包含于父窗体之中。

下面介绍多文档窗体的创建步骤, 实际上也就是 MDI 应用程序的创建步骤。

1. 创建父窗体

多文档窗体实际上是通过单文档窗体的属性 `FormStyle` 进行设置而得到的(不像 VC 那样从头来设置)。具体操作如下:

(1) 首先按照创建单文档窗体的方法创建一个文档窗体应用程序, 把程序命名为 `MDIApplication`。

(2) 为直观起见, 把窗体命名为 `MDIParent`, 即把窗体的 `Name` 属性设置为 `MDIParent`; 把窗体的 `FormStyle` 属性值设置为 `fsMDIForm`。`FormStyle` 属性还可以设置为其他值, 其意义可以参见 5.11.4 一节。

至此, 父窗体创建完毕。

2. 创建子窗体

创建子窗体需要用 Delphi 2005 IDE 重新创建一个 Form, 然后把它的 `FormStyle` 属性值设置为 `fsMDIChild`。具体方法如下。

(1) 在 Delphi 2005 IDE 中, 选择菜单命令 `File→New→Form`, 创建另一个 Form。

(2) 把新创建的 Form 的 `Name` 属性值设为 `MDIChild`, `FormStyle` 属性值设为 `fsMDIChild`。至此, 子窗体创建完毕。

在父窗体和子窗体创建完成以后, 就可以运行 `MDIApplication` 程序了, 运行结果如图 5.25 所示。

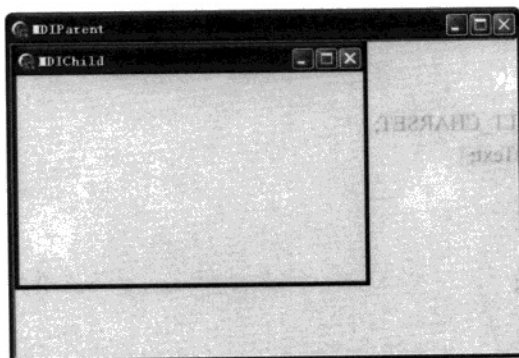


图 5.25 MDIApplication 程序的运行界面

多文档窗体由一个父窗体 MDIParent 和一个子窗体 MDIChild 组成。但是，在这个程序中不提供创建子窗体的功能，也就是说，在程序运行的时候父窗体自动打开了一个子窗体，运行了以后用户不能对其进行操作。

下面对该程序作进一步的改进，使得用户能够利用菜单创建多个子窗口（开始时没有子窗口）。

3. 改进 MDIApplication 程序

改进的基本思想是，把父窗体设置为应用程序的主窗体，同时创建子窗体类（而不是对象，避免在运行时有子窗体出现），在程序运行时利用菜单功能来创建子窗体的实例并显示。具体方法如下：

(1) 在 Delphi 2005 IDE 中，选择菜单命令 Project→Options...，将弹出 Project Options 对话框。

(2) 在 Project Options 对话框中，选择左边方框中的 Forms 结点，然后在出现的“Auto-create forms:”框中选择 MDIChild 选项（工程中创建的 Form 都会在这个框中出现），并单击 > 按钮，MDIChild 选项会移到“Available forms:”框中，结果如图 5.26 所示。

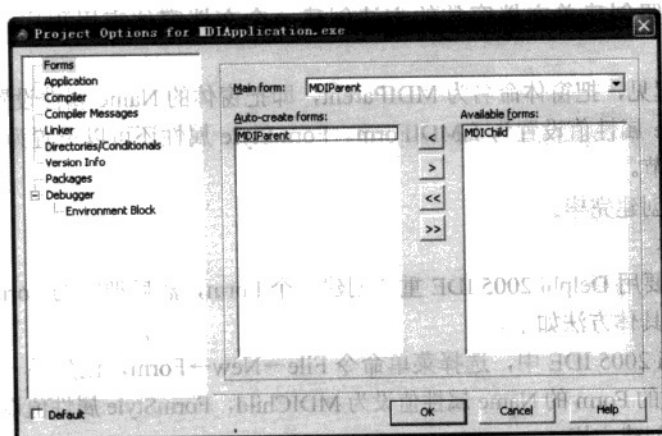


图 5.26 Project Options 对话框

(3) 单击 OK 按钮, 设置完毕。这个设置会使得在以后的运行中不会自动创建子窗体。

(4) 在窗体设计器中打开 MDIParent 窗体, 然后在窗体中添加主菜单组件 MainMenu, 并按照 5.7 节介绍的方法对菜单进行设计, 代码如图 5.27 所示。

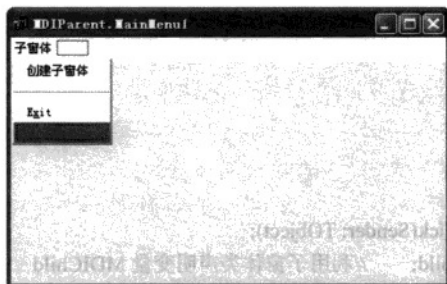


图 5.27 设计菜单

(5) 在代码编辑器中打开父窗体 MDIParent 对应的单元文件 Unit1.pas, 在 Interface 部分的 uses 编译指令中添加子窗体对应的单元文件名 Unit2, 代码如下:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menu, Unit2; // 添加了 Unit2, 用逗号隔开
```

(6) 双击主窗体上的菜单组件 MainMenu, 打开菜单设计器。然后双击“创建子窗体”标签, 将自动进入处理该菜单事件的函数的代码编辑处。修改该函数的代码如下:

```
procedure TMDIParent.N2Click(Sender: TObject);
var MDIChild: TMDIChild; // 利用子窗体类声明变量 MDIChild
begin
```

```
  MDIChild := TMDIChild.Create(Application);
  // 调用 TMDIChild 类的 Create 创建子窗体。
```

```
end;
```

这样, 父窗体对应的单元文件 Unit2.pas 的代码如下:

```
unit Unit2;
```

```
interface
```

```
uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menu, Unit2;
```

```
type
```

```
  TMDIParent = class(TForm)
```

```
    MainMenu1: TMainMenu;
```

```
    N1: TMenuItem;
```

```
    N2: TMenuItem;
```

```
    N3: TMenuItem;
```

```
    Exit1: TMenuItem;
```

```
    procedure N2Click(Sender: TObject);
```

```
  private
```

```

    { Private declarations }
public
    { Public declarations }
end;

var
    MDIParent: TMDIParent;
implementation
{$R *.dfm}

procedure TMDIParent.N2Click(Sender: TObject);
    var MDIChild: TMDIChild;    //利用子窗体类声明变量 MDIChild
begin
    MDIChild := TMDIChild.Create(Application);
    //调用 TMDIChild 类的 Create 创建子窗体
end;
end.

```

在 Unit1.pas 中引入了父窗体对应的单元文件 Unit2.pas, 该文件的代码如下:

```

unit Unit2;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs;

type
    TMDIChild = class(TForm)
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    MDIChild: TMDIChild;
implementation
{$R *.dfm}
end.

```

至此, MDIApplication 程序更改完毕。之后运行修改后的 MDIApplication 程序, 在出现的父窗体中选择菜单“子窗体”→“创建子窗体”命令, 就可以在父窗体中创建一个子窗体。连续四次使用该菜单命令创建子窗体, 结果如图 5.28 所示。

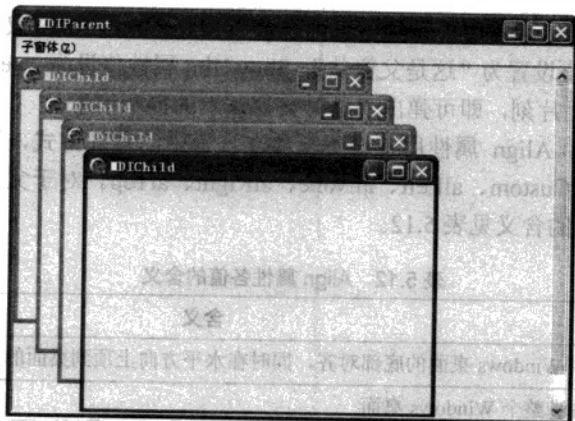


图 5.28 打开了四个子窗体的多文档窗体

5.11 熟悉 Delphi 窗体的主要属性

从上一节可以看到，父窗体是通过对普通窗体的属性进行设置而得到的。实际上，窗体的属性是很重要的，学会灵活地使用窗体属性，可以设计出许多风格各异的应用程序。为此，另辟一节专门讨论普通窗体属性的设置问题。

5.11.1 Action 标签页中的属性

所谓 Action 标签页是指在对象观察器 (Object Inspector) 中，单击 Action 项左边的“+”后展开的页面，如图 5.29 所示。对于观察器中的其他标签页也是如此。

(1) Caption 属性。很多组件都有这个属性，一般都是设置对象标题。窗体的 Caption 属性位于对象观察器的 Action 标签下。代码设置和访问格式为：

```
Form1.Caption := str;
str := Form1.Caption; //str 为字符串型, Form1 为窗体名称,
下同
```

(2) Enabled 属性。Enabled 属性用于设置窗体的有效性，其值为布尔型。Enabled 属性为 true (默认值) 时，窗体是有效的；如果为 false 则窗体是无效的。如果把父窗体的 Enabled 属性值设为 false，那么程序将“动不得”，需用 Windows 系统组合键命令——Ctrl+Alt+Del，打开 Windows 任务管理器来关闭它。代码设置和访问格式为：

```
Form1.Enabled := true; //或者 false
Bool := Form1.Enabled; //Bool 为布尔型
```

5.11.2 Help and Hints 标签和 Layout 标签中的属性

Help and Hints 标签中主要用的属性是 Hint 属性和 ShowHint 属性。它们用于创建窗体的

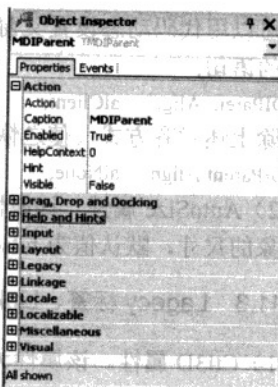


图 5.29 对象观察器中的 Action 标签页

提示信息, 其中, Hint 属性值用于设置信息内容, 当 ShowHint 属性值取为 true 时才有效。例如, 如果把 Hint 属性值设置为“这是父窗体”, ShowHint 属性值设为 true, 那么当程序运行后把鼠标停留在父窗体上片刻, 即可弹出“这时父窗体”的提示信息。

(1) Align 属性。Align 属性用于设置窗体在容器中的对齐方式, 该属性取值有七种: alBottom、alClient、alCustom、alLeft、alNone、alRight、alTop。对于父窗体 (其“容器”是 Windows 桌面), 各值的含义见表 5.12。

表 5.12 Align 属性各值的含义

属性值	含义
alBottom	靠 Windows 桌面的底部对齐, 同时在水平方向上顶到桌面的左右边
alClient	占满整个 Windows 桌面
alCustom	自定义方式, 由用户设定其大小
alLeft	靠桌面左边对齐, 同时在垂直方向上顶到桌面的上下边
alNone	没有用任何对齐方式
alRight	靠桌面右边对齐, 同时在垂直方向上顶到桌面的上下边
alTop	靠桌面的顶部对齐, 同时在水平方向上顶到桌面的左右边

也可以用代码动态设置父窗体的对齐方式, 如父窗体“使占满整个 Windows 桌面”, 可以用下列的语句:

```
MDIParent.Align := alClient;
```

解除上述对齐方式 (使之恢复到按设计尺寸显示) 则用下列语句:

```
MDIParent.Align := alNone;
```

(2) AutoSize 属性。该属性值为布尔型, 当它取值为 true 时, 自动把窗体调到刚能包含其中对象的尺寸, 默认值为 false。

5.11.3 Legacy 标签、Linkage 标签和 Localizable 标签中的属性

(1) Ctl3D 属性。该属性为布尔型, 当它取值为 true (默认值) 时, 窗体具有立体效果; 当取值为 false 时, 窗体呈平面效果。

(2) Menu 属性。Menu 属性用于设置窗体的主菜单, 其右边是一个下拉列表框。单击该下拉列表框将列出窗体中所包含的 MainMenu 组件对象的名称, 只要选择相应的菜单对象名称即可。一般来说, 窗体只有一个主菜单, 所以当在窗体中创建 MainMenu 组件对象时, 其名称自动成为该属性的值。


(3) Font 属性。Font 属性下面有 8 个子属性, 其中 Charset 属性用于设置窗体上所包含的全部文本字符 (其他组件上的字符也包含在内) 所采用的字符; Color 属性用于设置文本字符的颜色; Name 属性用于设置文本字符的字体; Height 属性用于设置字符的高度; Size 属性用于设置字符的字体大小; Style 属性则用于设置字符的风格, 比如是否用粗体、是否用斜体等。

利用窗体的 Font 属性可以对其包含的所有字符进行同时设置, 当然, 此后也可以单独对某些对象的属性进行设置。

(4) ClientHeight 属性和 ClientWidth 属性。ClientHeight 属性和 ClientWidth 属性分别用

于设置窗体中设计区域的高度和宽度，单位为像素。

(5) Height 属性和 Width 属性。Height 属性和 Width 属性分别用于设置窗体实际（外部）的高度和宽度，单位为像素。

(6) Icon 属性。每个应用程序标题栏的最左边都有一个图标，这个图标代表一定的意义。在 Delphi 2005 中利用 Icon 属性，用户可以加载自己的个性化图标，方法是：单击该属性右边的省略号按钮，将打开 Picture Editor 对话框，如图 5.30 所示。然后单击 Load 按钮，就可以加载已经编辑好的图标（.ico 文件）。

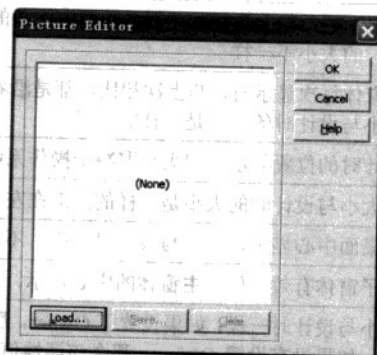


图 5.30 Picture Editor 对话框

(7) AlphaBlend 属性和 AlphaBlendValue 属性。AlphaBlend 属性和 AlphaBlendValue 属性一起用来设置窗体的透明度。当 AlphaBlend 属性为 true 时，AlphaBlendValue 属性值可以在 0 到 255 范围内变动。如果 AlphaBlendValue 属性的值为 0 则窗体完全透明，如果值为 255 则窗体不透明，如果介于 0 到 255 之间，那么窗体将呈现不同程度的透明。

5.11.4 Miscellaneous 标签中的属性

1. FormStyle 属性

FormStyle 属性的值将直接决定着 Delphi 应用程序的类型，它有四个值，即 fsMDIChild、fsMDIForm、fsNormal、fsStayOnTop。这些取值可以将 Delphi 应用程序界面分为两种类型：单文档界面(SDI, Single Document Interface)和多文档界面(MDI, Multiple Document Interface)。这些值的含义见表 5.13。

表 5.13 FormStyle 属性值的含义

属性值	含义
fsMDIChild	属于 MDI 窗体类型，此窗体是 MDI 窗体的子窗体
fsMDIForm	属于 MDI 窗体类型，此窗体是 MDI 窗体的父窗体
fsNormal	属于 SDI 窗体类型，此窗体是单文档界面 (SDI) 窗体或者是普通的对话框
fsStayOnTop	属于 SDI 窗体类型，但这种窗体总是位于其他窗体的前面。

2. Name 属性

Name 属性值是窗体的惟一标识符，该属性值只能在设计阶段设置，在运行时只能访问而

不能修改。程序要访问窗体必须通过这个属性。在工程开发中，与其他组件的命名一样，名称字符串必须具有一定的意义，至少有助于开发人员的记忆。但本书中的例子都不是有很多代码，所以往往都是采用默认值。

3. Position 属性

该属性用来设置窗体显示时的大小和位置，Position 属性的可能取值及其意义如表 5.14 所示。

表 5.14 Position 属性值及其含义

属性值	含义
poDefault	窗体每次显示时，与上次相比，都是沿着右下角的方向移动少许，但显示时的大小与设计时的大小不一样
poDefaultPosOnly	默认值，窗体每次显示时，与上次相比，都是沿着右下角的方向移动少许，但显示时的大小与设计时的大小是一样的
poDefaultSizeOnly	窗体以设计时的位置显示，但大小取决于操作系统窗口
poDesigned	显示时的大小与设计时的大小是一样的，但在左上角显示
poDesktopCenter	总是位于桌面中心显示，大小与设计时一样。不考虑多个监视器时的调整
poMainFormCenter	该属性对子窗体有效，位于主窗体的中心显示，大小与设计时一样
poOwnerFormCenter	显示时大小与设计时一样，如果设置了 Owner，那么窗体将在 Owner 指示的窗体中心显示；如果没有设置 Owner，那么该属性值的功能与 poMainFormCenter 属性值一样
poScreenCenter	总是位于屏幕中心显示，大小与设计时一样。考虑多个监视器时的调整

4. WindowState 属性

该属性用来设置窗体在运行时显示的状态，可能的取值有 wsNormal、wsMaximized、wsMinimized。当 WindowState 属性取值为 wsNormal 时，窗体将以设计时的大小显示；当取值为 wsMaximized 时，窗体将以最大化状态显示；如果为 wsMinimized 时，窗体将以最小化状态显示。

5.11.5 Visual 标签中的属性

1. BorderIcons 属性

BorderIcons 属性是一个集合类型的属性，用于设置窗体标题栏上图标、按钮等的显示。集合的元素包括 biSystemMenu、biMinimize、biMaximize、biHelp。BorderIcons 属性值可以有这四个元素中的若干个元素组成的集合，各元素的意见见表 5.15。

表 5.15 BorderIcons 属性值的含义

属性值	含义
biSystemMenu	如果 BorderIcons 属性值包含该值，则标题栏上将显示系统菜单；如果没有包含该值，那么不但没有系统菜单，而且 biMinimize、biMaximize、biHelp 的设置都将无效
biMinimize	如包含该值，标题栏右边将出现最小化按钮
biMaximize	如包含该值，标题栏右边将出现最大化按钮
biHelp	如包含该值，标题栏右边将出现帮助按钮

2. BorderStyle 属性

BorderStyle 属性用于设置窗体的边框, 其可能的取值及各值的含义如表 5.16 所示。

表 5.16 BorderStyle 属性值的含义

属性值	含义
bsDialog	有默认的粗边框, 但大小不能改变、标题栏上没有图标、最大化和最小化按钮, 只有一个关闭按钮
bsNone	没有边框 (包括标题栏)
bsSingle	窗体边框比较细, 大小不能改变
bsSizeable	默认值, 有边框, 可以动态调整窗体大小
bsSizeToolWin	与 bsDialog 相似, 但标题变小, 窗体大小可以改变
bsToolWindow	与 bsSizeToolWin 相似, 但不能改变窗体的大小

3. Color 属性

Color 属性用于设置窗体背景色。

5.12 创建与设计 Delphi MDI 应用程序的主菜单

为了很好地说明问题, 首先按照 5.10.2 介绍的方法创建一个工程名为 MutliMenuMDI 的多文档窗体应用程序, 其中父窗体句柄为 MDIParent, 对应的单元文件名为 Unit1.pas, 其相应的窗体文件名为 Unit1.dfm; 子窗体的句柄为 MDIChild, 对应的单元文件名为 Unit2.pas, 其相应的窗体文件名为 Unit2.dfm。并按照 5.10.2 节介绍的方法把父窗体设为应用程序的主窗体。

MutliMenuMDI 程序的工程结构如图 5.31 所示。

在窗体中创建菜单的方法在前面已经有所涉及, 为完整起见, 本节将详细而系统地介绍在父窗体创建主菜单和在子窗体中创建主菜单的一般方法。

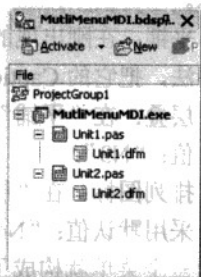


图 5.31 MutliMenuMDI 程序的工程

5.12.1 创建父窗体中的主菜单

在父窗体中创建主菜单的基本方法可按照下列步骤进行:

(1) 打开 MutliMenuMDI 程序的主窗体 (父窗体), 方法是: 在工程管理器中双击结点 Unit1.dfm, 然后在工具面板的 Standard 标签页中找到 MainMenu 组件, 并把它拖到父窗体中, 如图 5.32 所示。

(2) 双击 MainMenu 组件, 将打开主菜单设计器。选择第一个菜单项, 把它的 Caption 属性值设为 “子窗口管理(&M)”, name 属性值设为 NChildWinManaging。然后设置以下子菜单项:

1) 创建子窗口。当菜单项 NChildWinManaging 被设置的时候, 其下面就已经出现了一个空的子菜单项。当菜单项 NChildWinManaging 设置完毕后, 可以直接单击选中其下出现的空

菜单项，并在其属性 Caption 栏输入“创建子窗口”，Name 属性值设置为 NNewWin。同时其下面也自动出现一个空的子菜单项。

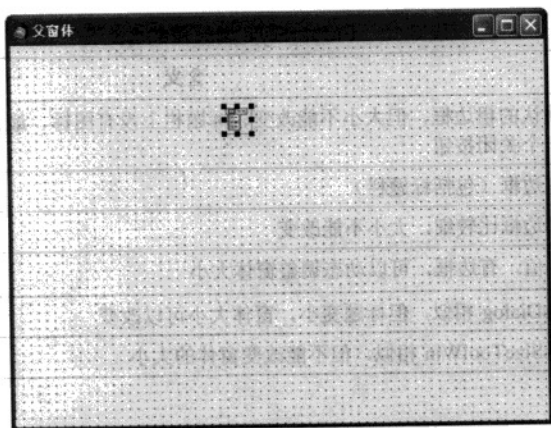


图 5.32 在父窗体创建 TMainMenu 类对象

2) 窗口布局。单击选中 NNewWin 子菜单项下的空子菜单项，把该菜单的 Caption 属性值设为“窗口布局”，Name 属性值设为 NWinLayout。然后单击“窗口布局”栏，在弹出的菜单中选择 Create Submenu 菜单项，“窗口布局”栏的右边出现箭头符号，并出现一个空的子菜单，如图 5.33 所示。

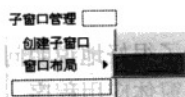


图 5.33 创建子菜单

- 平铺：在刚出现的空的子菜单中按照上述类似的设置方法，把它的 Caption 属性值设为“平铺”，Name 属性采用默认值“N2”。
- 层叠：在“平铺”子菜单下面设置“层叠”子菜单，方法同上，Name 属性采用默认值：“N3”。
- 排列图标：在“层叠”子菜单下面设置“排列图标”子菜单，方法同上，Name 属性采用默认值：“N4”。

以上三个菜单就构成了子菜单“窗口布局”的子菜单。

3) 关闭当前窗口。回到“窗口布局”下的空子菜单，把它的 Caption 属性值设为“关闭当前窗口”，Name 属性值设为 NCloseChildWin。

4) 关闭所有的子窗口。选择“关闭当前窗口”下的空子菜单，把它的 Caption 属性值设为“关闭所有的子窗口”，Name 属性值设为 NCloseAllChildWin。

5) 分隔符。选择“关闭所有的子窗口”下的空子菜单，把它的 Caption 属性值设为“-”，即可形成菜单中的一条分隔符。

6) 关闭主窗体。选择“关闭所有的子窗口”下的空子菜单，把它的 Caption 属性值设为“关闭主窗体”，Name 属性值设为 NCloseSelf。

选择“子窗口管理”菜单项右边的空白菜单项，把该空白菜单项的 Caption 属性值设为“帮助(&H)”，Name 属性值设为 NHelp。可以按照上述的方法，进一步设置“帮助”菜单下的各项子菜单（如果需要的话）。

经过以上设置，就可以得到主窗体的菜单，其结构如图 5.34 所示。

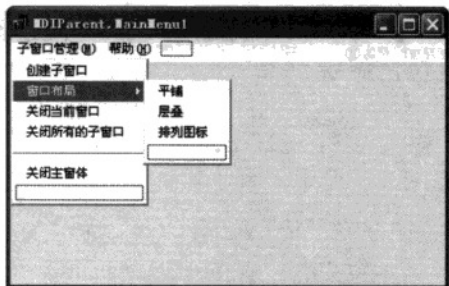


图 5.34 主窗体的菜单结构

如果需要删除某一菜单项，则在该菜单项上右击，在弹出的快捷菜单中选择 Delete 命令即可，如果要插入菜单项则选择 Insert 命令。

5.12.2 创建子窗体中的主菜单

在子窗体中创建主菜单的方法与在父窗体中创建主菜单的方法完全一样。首先在设计器中打开子窗体，并把 MainMenu 组件拖到子窗体中，然后双击 MainMenu 组件打开菜单设计器。按照上面介绍的方法进行设计，结果如图 5.35 所示。

为了创建一个比较有意义的例子，进一步加入 Memo 组件，以制作一个简单的多文档编辑器。Memo 组件的属性值设置如下：

(1) Name。采用默认值：Memo1。

(2) Align。设置为 alClient，以使 Memo 对象充满整个子窗体。

此外，多文档编辑器还必须有打开文件和保存文件的功能，所以还必须在子窗体中加入 OpenFileDialog 和 SaveDialog 属性。结果如图 5.36 所示。

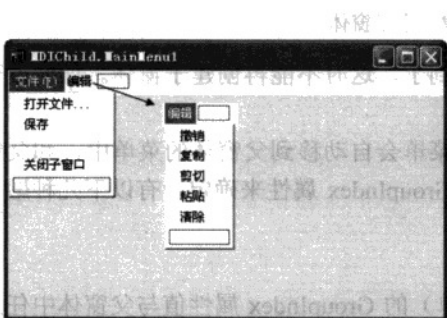


图 5.35 菜单设计结果

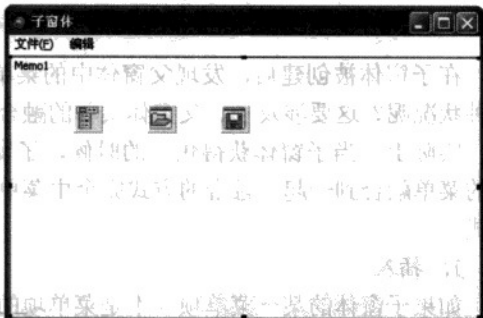


图 5.36 子窗体的设计结果

5.12.3 父窗体和子窗体中菜单的融合

经过以上对父窗体和子窗体进行的设置，可以运行 MutliMenuMDI 程序了，运行结果如图 5.37 所示。

在运行界面中，选择“子窗口管理”→“创建子窗口”命令，将在父窗体中创建一个子窗体，如图 5.38 所示。

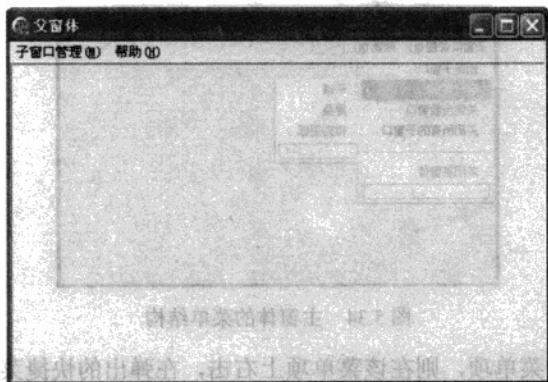


图 5.37 MutliMenuMDI 程序的运行界面

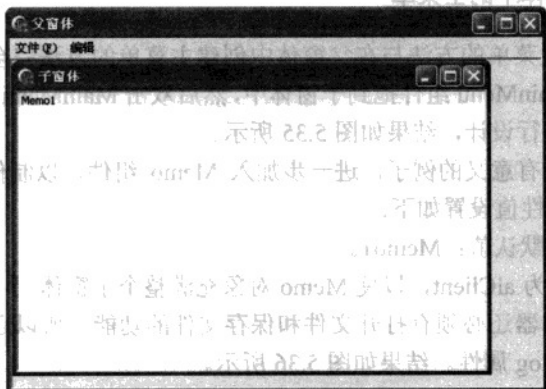


图 5.38 在父窗体中创建一个子窗体

在子窗体被创建后，发现父窗体中的菜单找不到了，这时不能再创建子窗体。如何改变这种状况呢？这要涉及子、父窗体菜单的融合问题。

实际上，当子窗体获得焦点的时候，子窗体的菜单会自动移到父窗体的菜单中，和父窗体的菜单融合到一起。融合的方式完全由菜单项的 `GroupIndex` 属性来确定，有以下几种融合方式。

1. 插入

如果子窗体的某一菜单项（不是菜单项的子菜单）的 `GroupIndex` 属性值与父窗体中任意菜单项的 `GroupIndex` 属性值均不相同，则在菜单融合时该菜单项（连同它的子菜单）将插入到父窗体菜单列表中的相应位置（注意，窗体菜单列表中菜单项的位置取决于菜单项的 `GroupIndex` 属性值。如果该属性值越小，则相应菜单项就越靠前；如果该属性值越大，则相应菜单项就越靠后）。

2. 替换

如果子窗体的某一菜单项的 `GroupIndex` 属性值与父窗体中的某一个或某一些菜单项的 `GroupIndex` 属性值相同，那么在菜单融合时该菜单项替换父窗体中所有与之 `GroupIndex` 属性值相同的菜单项。

注意, 在一个窗体中各菜单项的 `GroupIndex` 属性值可以相同, 这时不会导致相互替换。一般地, 靠前的菜单项的 `GroupIndex` 属性值应小于或等于其后的菜单项的 `GroupIndex` 属性值; 如果强行把前面的属性值改大, 以至于大于其后菜单项的 `GroupIndex` 属性值, 那么其后面菜单项的 `GroupIndex` 属性值也将自动变成与之相等。

菜单项的 `GroupIndex` 属性的默认值为 0。在上面创建父窗体和子窗体的时候, 都没有设置该属性值, 而是采用默认值 0。所以, 父窗体中的菜单项和子窗体中的菜单项的 `GroupIndex` 属性值都是相同的 (同为 0)。因此, 在父窗体中创建子窗体而使子窗体获得焦点时, 子窗体中的菜单项就全部替换了父窗体中的菜单项, 这就是 `MutliMenuMDI` 程序运行并创建子窗体后父窗体菜单找不到的原因。

如果在创建子窗体后, 要使得父窗体的菜单不被替换, 并以“文件(E)→子窗口管理(M)→编辑→帮助(H)”的顺序排列, 那么可以在菜单编辑器中设置各菜单项的 `GroupIndex` 属性值, 结果如表 5.17 所示。

表 5.17 子、父窗体中各菜单项 `GroupIndex` 属性值的设置

菜单项	所属窗体	<code>GroupIndex</code> 属性值
文件(E)	子窗体	0
子窗口管理(M)	父窗体	1
编辑	子窗体	2
帮助(H)	父窗体	3

经过以上设置后, `MutliMenuMDI` 程序运行时可以创建多个子窗体, 且子窗体中的菜单项按照预计的设计插入到了父窗体的菜单列表中, 结果如图 5.39 所示。

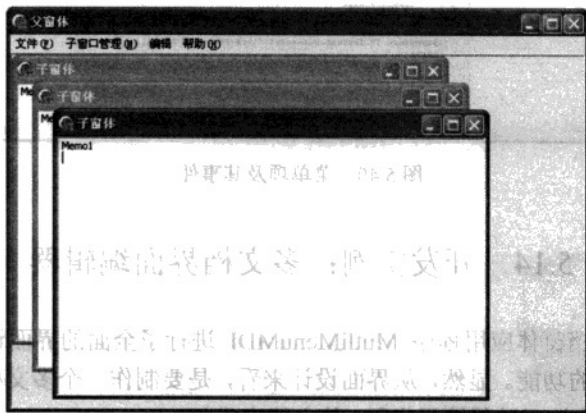


图 5.39 `MutliMenuMDI` 程序运行结果

5.13 熟悉 Delphi 的菜单事件

一个菜单项可以触发六个事件, 分别为 `Action`、`OnAdvanced`、`OnDrawItem`、`OnClick`、`OnDrawItem`、`SubMenuImages` 事件。其中最常用的是 `OnClick` 事件。选择任一菜单项, 都会

触发其相应的 OnClick 事件,这也是在编程中用得最多的事件。

编写事件处理程序代码的方法是:双击相应的(子)菜单项,将自动创建处理事件过程的框架并进入到代码编辑的地方。在这种方式下,过程名采用默认值。如果要更改过程名或先编写过程名再在进入过程体中写代码,则要选择相应的菜单项,然后选择(子)菜单项(图 5.40 中选择了“创建子窗口”子菜单项)并打开对象观察器,单击选中事件选项卡,找到 OnClick 方法并在其右边输入过程名(图 5.40 中输入了“NNewWinClick”),最后双击过程名即可进入或创建过程框架并进入到代码编辑的地方。在编辑处就可以编写处理事件的代码了。在图 5.40 中,在双击了过程名 NNewWinClick 后即可产生如下的过程框架:

```
procedure TMDIParent.NNewWinClick(Sender: TObject);
begin
end;
```

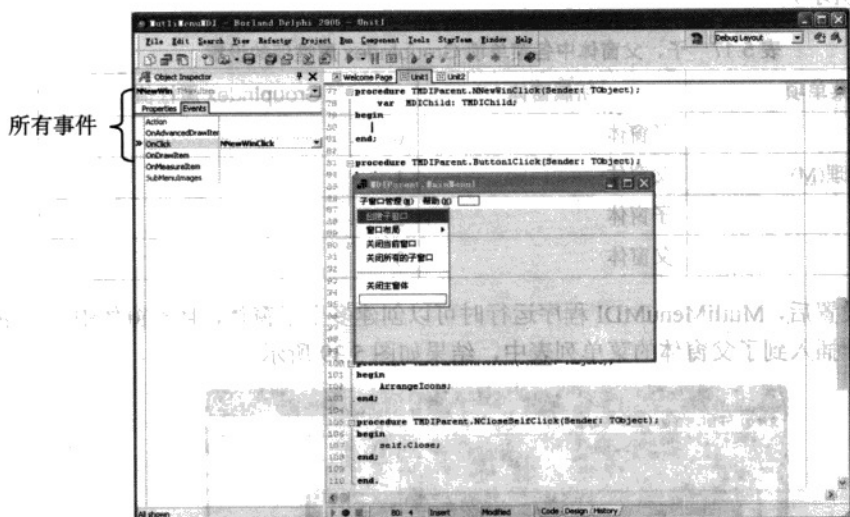


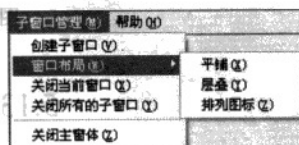
图 5.40 菜单项及其事件

5.14 开发实例：多文档界面编辑器

以上虽然对多文档窗体应用程序 MultiMenuMDI 进行了全面的界面设计,但是该程序到目前为止还没有任何的功能。显然,从界面设计来看,是要制作一个多文档界面的编辑器。为此,必须为各个事件的处理过程编写代码,使之能完成相应的功能。

5.14.1 实现“子窗口管理”菜单项功能

“子窗口管理”是父窗体的一项主要功能,它包含的子功能如图 5.41 所示。



下面介绍各子功能的实现方法。

1. 创建子窗口

该项菜单项的实现方法已经在 5.2.2 节中作过介绍，为完整起见，在这里也作了简单的介绍。双击父窗体中的 MainMenu 组件，打开菜单设计器，然后选择“创建子窗口”，并打开对象观察器，选择其中的事件选项卡，找到 OnClick 项；在 OnClick 项右边的文本框中输入 NNewWinClick 作为过程名，最后双击该文本框即可自动生成事件处理过程的代码框架，同时鼠标自动进入代码编辑处；在此处写入相应的代码，最后得到如下的过程代码：

```
procedure TMDIParent.NNewWinClick(Sender: TObject);  
    var MDIChild: TMDIChild;  
begin  
    MDIChild := TMDIChild.Create(Application); //创建子窗体  
end;
```

同时记住要在单元文件的 uses 部分加入字符串“Unit2”，否则编译时 TMDIChild 找不到而出错。

2. 窗口布局

“窗口布局”子菜单又包含三个菜单项：平铺、层叠和排列图标。这三个菜单项的实现方法与“创建子窗口”的实现方法一样，最后得到相应的过程代码如下：

```
procedure TMDIParent.Button1Click(Sender: TObject);  
begin  
    Cascade; //层叠子窗口  
end;
```

```
procedure TMDIParent.N2Click(Sender: TObject);  
begin  
    Tile; //平铺  
end;
```

```
procedure TMDIParent.N3Click(Sender: TObject);  
begin  
    Cascade; //层叠  
end;
```

3. 关闭当前窗口

用类似以上的方法创建“关闭当前窗口”菜单项对应的过程，其代码如下：

```
procedure TMDIParent.NCloseChildWinClick(Sender: TObject);  
begin  
    ActiveMDIChild.Close; //关闭当前活动的子窗体  
end;
```

4. 关闭所有的子窗口

用类似以上的方法创建“关闭所有的子窗口”菜单项对应的过程，其代码如下：

```
procedure TMDIParent.NCloseAllChildWinClick(Sender: TObject);  
var n: integer;  
begin  
    for n:=0 to MDIChildCount-1 do //关闭所有的子窗体  
        MDIChildren[n].Close;
```

end;

5. 关闭主窗体

用类似以上的方法创建“关闭主窗体”菜单项对应的过程，其代码如下：

```
procedure TMDIParent.NCloseSelfClick(Sender: TObject);
```

```
begin
```

```
  self.Close;
```

```
end;
```

5.14.2 实现“文件”菜单项功能

“文件”菜单位于子窗体的菜单列表中，它包含三个菜单项，如图 5.42 所示。

文件打开和保存的方法在第 3 章已经介绍了，在此只给出相应过程的代码。

1. 打开文件

“打开文件”菜单项对应的过程代码如下：

```
procedure TMDIChild.NOpenfileClick(Sender: TObject);
```

```
begin
```

```
  OpenDialog1.Filter := 'Txt Files(*.txt)*.txt'; //仅打开文本文件
```

```
  if OpenDialog1.Execute() then
```

```
  begin
```

```
    Memo1.Clear;
```

```
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

```
  end;
```

```
end;
```

2. 保存

“保存”菜单项对应的过程代码如下：

```
procedure TMDIChild.NSavefileClick(Sender: TObject);
```

```
begin
```

```
  SaveDialog1.Filter := 'Word files(*.doc)*.doc|Txtfile(*.txt)*.txt';
```

```
  //可以保存为 doc 文件或 txt 文件
```

```
  SaveDialog1.FilterIndex := 2; //设置默认显示的文件类型为*.txt 文件
```

```
  if SaveDialog1.Execute() then
```

```
  begin
```

```
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
```

```
  end;
```

```
end;
```

3. 关闭子窗口

“关闭子窗口”菜单项对应的过程代码如下：

```
procedure TMDIChild.NCloseChildClick(Sender: TObject);
```

```
begin
```

```
  self.Close;
```

```
end;
```

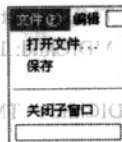


图 5.42 “文件”菜单的菜单项

5.14.3 实现“编辑”菜单功能

“编辑”菜单位于子窗体的菜单列表中，它包含五个菜单项，如图 5.43 所示。

其中，“撤销”、“复制”、“剪切”、“粘贴”、“清除”等这 5 个菜单项对应的功能都是利用 TMeme 类提供的方法来实现的，其相应的代码语句如下：

```

Memo1.Undo;
Memo1.CopyToClipboard;
Memo1.CutToClipboard;
Memo1.PasteFromClipboard;
Memo1.Clear;

```

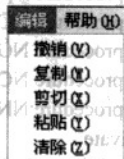


图 5.43 “编辑”菜单的菜单项

5.14.4 执行多文档窗体应用程序

为了使每次创建的子窗体都显示不同的标题信息，如显示“子窗体 1”，“子窗体 2”，…，在单元文件中定义一个全局变量：

```
ChildNum: Integer;
```

另外，还在 TMDIParent.NNewWinClick 过程中的最后一行加入下列代码：

```
MDIChild.Caption := MDIChild.Caption+IntToStr(ChildNum);
```

该语句用于更改窗口的标题。

经过以上设计和代码编写过程以后，MutliMenuMDI 程序的单元文件 Unit2.pas 的全部代码如下：

```

unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls;

type
  TMDIChild = class(TForm)
    MainMenu1: TMainMenu;
    NFileChild: TMenuItem;
    NEdit: TMenuItem;
    NCopy: TMenuItem;
    N1: TMenuItem;
    NCloseChild: TMenuItem;
    NOpenfile: TMenuItem;
    Memo1: TMemo;
    OpenDialog1: TOpenDialog;
    NSavefile: TMenuItem;
    Npaste: TMenuItem;
    NUndo: TMenuItem;
    NClear: TMenuItem;
    NCut: TMenuItem;
    SaveDialog1: TSaveDialog;

```

```

    procedure NSavefileClick(Sender: TObject);
    procedure NCutClick(Sender: TObject);
    procedure NClearClick(Sender: TObject);
    procedure NUndoClick(Sender: TObject);
    procedure NpasteClick(Sender: TObject);
    procedure NCopyClick(Sender: TObject);
    procedure NOpenfileClick(Sender: TObject);
    procedure NCloseChildClick(Sender: TObject);
    procedure NNewChildClick(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    MDIChild: TMDIChild;
    ChildNum: integer;
implementation
{$R *.dfm}

procedure TMDIChild.NNewChildClick(Sender: TObject);
begin
    ShowMessage('111');
end;

procedure TMDIChild.NCloseChildClick(Sender: TObject);
begin
    self.Close;
end;

procedure TMDIChild.NOpenfileClick(Sender: TObject);
begin
    OpenFileDialog1.Filter := 'Txt Files(*.txt)*.txt'; //仅打开文本文件
    if OpenFileDialog1.Execute() then
    begin
        Memo1.Clear;
        Memo1.Lines.LoadFromFile(OpenFileDialog1.FileName);
    end;
end;

procedure TMDIChild.NCopyClick(Sender: TObject);
begin
    Memo1.CopyToClipboard;
end;

procedure TMDIChild.NpasteClick(Sender: TObject);
begin
    Memo1.PasteFromClipboard;
end;

```

```

procedure TMDIChild.NUndoClick(Sender: TObject);
begin
    Memo1.Undo;
end;

procedure TMDIChild.NClearClick(Sender: TObject);
begin
    Memo1.Clear;
end;

procedure TMDIChild.NCutClick(Sender: TObject);
begin
    Memo1.CutToClipboard;
end;

procedure TMDIChild.NSavefileClick(Sender: TObject);
begin
    SaveDialog1.Filter := 'Word files(*.doc)*.doc|Textfile(*.txt)*.txt';
    //可以保存为 doc 文件或 txt 文件
    SaveDialog1.FilterIndex := 2; //设置默认显示的文件类型为*.txt 文件
    if SaveDialog1.Execute() then
    begin
        Memo1.Lines.SaveToFile(SaveDialog1.FileName);
    end;
end;
end.

```

代码编写完毕以后，编译、执行该程序。执行后创建了三个子窗体并打开了三个文件，结果如图 5.44 所示。其中，各个菜单都可以发挥出既定的作用，实现了相应的功能。

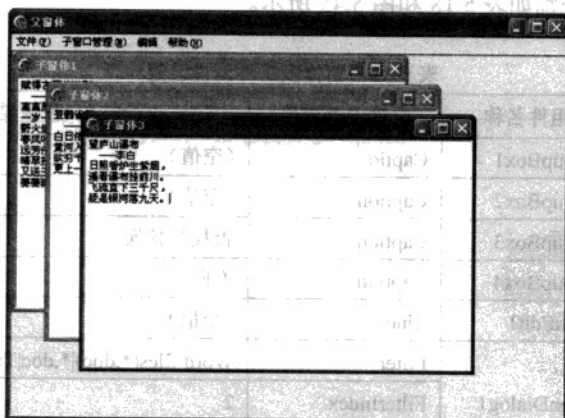


图 5.44 多文档窗体应用程序 MultiMenuMDI 的执行结果（打开了三个文本文件）

5.15 开发实例：具有查找和替换功能的 RTF 编辑器

在 5.5.7 节中已经介绍了文本编辑器的制作，但其目的是为了熟悉对话框组件的使用方法。

在本节的实例开发中，也将创建一个编辑器——RTF 编辑器，它不但具有打开、保存和字体设置等功能，还具有不同方向（向上和向下）的文本替换和查找功能。查找和替换功能完全可以用组件 TFindDialog 和 TReplaceDialog 来实现，但出于学习目的，在本实例中将用自己编写的代码来实现。在代码的实现过程中，融入了许多编程技巧和经验，相信读者通过对本实例的学习，将对基于字符串的编程技术有一个实质性的提高。

5.15.1 功能设计

本实例是一个用 TRichEdit 组件实现的 RTF 编辑器，其主要包括以下功能。

- 可以打开和保存文本文件，也可以在编辑器中直接输入字符，然后保存为文本文件。
- 设置文本的字体，可以设置编辑器的背景颜色。
- 查找输入的字符，并且可以从向上或向下的方向上找。
- 替换输入的字符，并且可以从向上或向下的方向上进行替换。

5.15.2 程序创建与界面设计

创建一个 VCL Forms 应用程序，命名为 RTFEdit.dpr，其窗体单元文件命名为 RTFEditUnit.pas。然后在窗体上添加四个 GroupBox 组件，用于对其他组件进行分组；添加一个 RichEdit 组件，用于显示文本；添加四个对话框组件：OpenDialog、SaveDialog、FontDialog 和 ColorDialog，它们分别用于打开和保存文本文件，以及设置编辑器的字体颜色和编辑器的背景颜色；添加两个 Edit 组件，用于输入查找字符串和替换字符串；添加一个 CheckBox 组件，用于确定查找匹配时是否区分大小写；添加两个 RadioButton 组件，用于确定搜索的方向；添加六个 Button 组件，以用于打开文件、保存文件、字体设置、颜色设置、查找下一个和替换等操作。最后添加两个 Label 组件用于标注文本框的含义，同时设置这些组件的属性，调整组件位置和大小，结果分别如表 5.18 和图 5.45 所示。

表 5.18 窗体及组件属性设置

组件类型	组件名称	属性设置项目	设置结果
GroupBox	GroupBox1	Caption	(空值)
	GroupBox2	Caption	(空值)
	GroupBox3	Caption	查找与替换
	GroupBox4	Caption	方向
RichEdit	RichEdit1	Lines	(空值)
OpenDialog	OpenDialog1	Filter	Word files(*.doc) *.doc Txtfile(*.txt) *.txt
		FilterIndex	2
		Options	[ofHideReadOnly,ofEnableSizing]
SaveDialog	SaveDialog1	Filter	(空值，用代码设置)
		FilterIndex	(空值，用代码设置)
		Options	[ofOverwritePrompt,ofHideReadOnly,ofEnableSizing]

续表

组件类型	组件名称	属性设置项目	设置结果
FontDialog	FontDialog1		(所有属性采用默认值)
ColorDialog	ColorDialog1		(所有属性采用默认值)
Button	OpenFile	Caption	打开文件
	SaveFile	Caption	保存文件
	SetFont	Caption	字体设置
	SetColor	Caption	颜色设置
	Find	Caption	查找下一个
	Replace	Caption	替 换
Edit	tofindText	Text	(空值)
	toreplaceText	Text	(空值)
CheckBox	IsCase	Caption	区分大小写
RadioButton	upRadio	Caption	向上
	downRadio	Caption	向下
Label	Label3	Caption	查找内容:
	Label4	Caption	替 换 为:
Form	Form1	Caption	RTF 编辑器



图 5.45 程序 RTFEdit 的设计界面

5.15.3 主要步骤和核心代码的实现

本实例具有查找、替换文本，以及打开、编辑和保存文件等多种功能，这些功能要分别予以实现。以下分别讲述。

1. TForm1 类成员函数 downfindStr

该函数用于在编辑器中从 startPos 位置开始向下搜索由参数 tofind 给定的子串。如果搜索成功，则把搜索的子串处于选定状态；如果没有搜索到，则给出相应的提示信息。函数代码如下：

```

procedure TForm1.downfindStr(startPos:integer; tofind:string);
//从 startPos 开始向下搜索子串 tofind
var str,nrstr:string;
    findpos,len,nrpos : integer;

begin
    str := RichEdit1.Text;
    if isCase.Checked = false then str := ansilowercase(str); //不区分大小写
    len := length(str);
    str := copy(str, startPos+1);
    findpos := pos(tofind,str);
    nrstr := copy(str,1,findpos+length(tofind)-2);
    if pos(nr,nrstr) = 0 then
    //匹配行中的字符（从 startPos 位置起，没有回车换行符）
    begin
        flag := 0;
        if findpos <> 0 then
        begin
            RichEdit1.SelStart := (len-length(str)) + findpos - 1;
            RichEdit1.SelLength := length(tofind);
            findNum := findNum + 1; //统计找到子串的数量
            flag := 1;
        end
        else
        begin
            if findNum <> 0 then showmessage('搜索完成，一共有 '+IntToStr(findNum)+'
                ' 处出现搜索项')
            else showmessage('搜索完毕，未找到搜索项！');
            findNum := 0;
        end;
    end
    else
        //处理回车、换行符
    begin
        flag := 0;
        nrpos := pos(nr,nrstr);
        delete(nrstr,nrpos,2); //在 nrstr 中删回车、换行符
        findpos := pos(tofind,nrstr);
        if findpos <> 0 then
        begin
            RichEdit1.SelStart := startPos+findpos-1; //(len-length(nrstr)-2) + findpos - 1;
            RichEdit1.SelLength := length(tofind)+2;
            findNum := findNum + 1; //统计找到子串的数量
            flag := 1;
        end
    end
end;

```

```

end
else
begin
    RichEdit1.SelStart := RichEdit1.SelStart + 2;
    flag := 0;
    str := RichEdit1.Text;
    len := length(str);
    str := copy(str, RichEdit1.SelStart+1);
    findpos := pos(tofind, str);
    if findpos <> 0 then
    begin
        RichEdit1.SelStart := (len-length(str)) + findpos - 1;
        RichEdit1.SelLength := length(tofind);
        findNum := findNum + 1;    //统计找到子串的数量
        flag := 1;
    end
    else
    begin
        if findNum < 0 then
            showmessage('搜索完成，一共有 '+IntToStr(findNum)+' 处出现搜索项')
        else showmessage('搜索完毕，未找到搜索项! ');
        findNum := 0;
    end;
    end;
end;
end;
end;

```

2. TForm1 类成员函数 upfindStr

该函数与函数 downfindStr 的功能基本一样，只是搜索的方向不一样，downfindStr 是向下搜索，而 upfindStr 则是向上搜索。其代码如下：

```

procedure TForm1.upfindStr(startPos: integer; tofind: string);
    //从 startPos 开始向上搜索子串 tofind
var str, curStr: string;
    findpos, len, i: integer;

begin
    i := startPos-length(tofind)+1;
    str := RichEdit1.Text;
    if isCase.Checked = false then str := ansilowercase(str);    //不区分大小写
    str := copy(str, 1, startPos);
    while i >= 1 do
    begin

        if nrflag=0 then
        begin
            curStr := copy(str, i, length(tofind));
            nrflag := pos(nr, curStr);
            if nrflag <> 0 then    //delete(curStr, pos(nr, curStr), 2);    //删除回车符和换行符

```

```

begin
    i:=i-1;
    continue;
end;
end
else //if nrflag<>0 then
begin
    curStr := copy(str,i,length(tofind)+2);
    nrflag := pos(nr,copy(curStr,1,length(tofind)));
    delete(curStr,pos(nr,curStr),2);           //删除回车符和换行符
end;
if ansicomparestr(curStr,tofind) = 0 then
begin
    RichEdit1.SelStart := i-1;
    if nrflag<>0 then RichEdit1.SelLength := length(tofind)+2
    else RichEdit1.SelLength := length(tofind);
    findNum := findNum + 1;
    if nrflag=0 then curPos := RichEdit1.SelStart+length(tofind)-1
    else curPos := RichEdit1.SelStart+length(tofind)-1+2;
    exit;
end;
i := i - 1;
end;
if i<1 then
begin
    if findNum <> 0 then
        showmessage('搜索完成, 一共有 '+IntToStr(findNum)+' 处出现搜索项')
    else showmessage('搜索完毕, 未找到搜索项! ');
    findNum := 0;
end;
end;
end;

```

3. “查找下一个”按钮单击事件处理程序

当单击“查找下一个”按钮时, 将在编辑器中搜索编辑框 tofindText 输入的字符串, 并根据复选框 IsCase 确定是否区分大小写, 以及根据单选框 downRadio 和 upRadio 来确定搜索的方向 (通过调用函数 downfindStr 或函数 upfindStr 来实现)。该处理程序代码如下:

```

procedure TForm1.FindClick(Sender: TObject);
var    i,curCursorPos: integer;
        tofind : string;

begin
    tofind := tofindText.Text;
    if isCase.Checked = false then tofind := ansilowercase(tofind); //不区分大小写
    if downRadio.Checked then
    begin
        if flag=1 then RichEdit1.SelStart := RichEdit1.SelStart + 1;
        curCursorPos := RichEdit1.SelStart;
    end;

```

```

        downfindStr(curCursorPos,tofind);
    end;
    if upRadio.Checked then
    begin
        if curPos=0 then
        begin
            curPos := RichEdit1.SelStart;
        end;
        curCursorPos := curPos;
        upfindStr(curCursorPos,tofind);
    end;
end;

```

4. “替换”按钮单击事件处理程序

当单击“查找下一个”按钮时，将在编辑器中搜索编辑框 tofindText 输入的字符串。如果搜索到，则提示是否把它替换为编辑框 toreplaceText 输入的字符串，并由用户根据需要做出选择；如果搜索不到，则给出相应的提示信息。搜索的方向以及匹配方式与查找功能一样。该处理程序代码如下：

```

procedure TForm1.ReplaceClick(Sender: TObject);
var tofind:string;
    curCursorPos: integer;

begin
    tofind := tofindText.Text;
    if isCase.Checked = false then tofind := ansilowercase(tofind); //不区分大小写
    if downRadio.Checked then
    begin
        if flag=1 then RichEdit1.SelStart := RichEdit1.SelStart + 1; //向下替换时
        curCursorPos := RichEdit1.SelStart;
        downfindStr(curCursorPos,tofind);
        if RichEdit1.SelLength <> 0 then //找到子串
        begin
            if MessageDlg('要替换当前子串吗?',mtConfirmation,[mbYes,mbNo],0)=mrYes then
            begin
                RichEdit1.SelText := toreplaceText.Text;
            end;
        end;
    end;
    if upRadio.Checked then
    begin
        if curPos=0 then
        begin
            curPos := RichEdit1.SelStart; //向上搜索时
        end;
        curCursorPos := curPos;
        upfindStr(curCursorPos,tofind);
    end;
end;

```

```

if RichEdit1.SelLength <> 0 then //找到子串
begin
    if MessageDlg('要替换当前子串吗?', mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    begin
        RichEdit1.SelText := toreplaceText.Text;
    end;
end;
end;
end;

```

5. “打开文件”、“保存文件”、“字体设置”和“颜色设置”按钮单击事件处理程序

由于使用现成的组件，所以这几个按钮对应的实现代码比较简单，在此不作详细分析。它们对应的事件处理程序代码分别如下：

// 【打开文件】

```

procedure TForm1.OpenFileClick(Sender: TObject);
begin
    if OpenFileDialog1.Execute() then
    begin
        RichEdit1.Clear;
        RichEdit1.Lines.LoadFromFile(OpenDialog1.FileName);
    end;
end;

```

// 【保存文件】

```

procedure TForm1.SaveFileClick(Sender: TObject);
begin
    SaveDialog1.Filter := 'Word files(*.doc)|*.doc|Txtfile(*.txt)|*.txt';
    SaveDialog1.FilterIndex := 2; //设置默认显示的文件类型为*.txt 文件
    if SaveDialog1.Execute() then
    begin
        RichEdit1.Lines.SaveToFile(SaveDialog1.FileName);
    end;
end;

```

// 【字体设置】

```

procedure TForm1.SetFontClick(Sender: TObject);
begin
    if FontDialog1.Execute() then
    begin
        RichEdit1.Font := FontDialog1.Font;
    end;
end;

```

// 【颜色设置】

```

procedure TForm1.SetColorClick(Sender: TObject);
begin
    if ColorDialog1.Execute() then

```

```

begin
    RichEdit1.Color := ColorDialog1.Color;
end;
end;

```

上面是本实例的核心代码。除此之外，还要定义相关的全局变量，并在窗体的 OnCreate 事件处理程序 FormCreate 中对它们进行初始化：

```

var
    Form1: TForm1;
    curPos, findNum, flag, nrflag: integer;
    nr: string;

procedure TForm1.FormCreate(Sender: TObject);
begin

```

```

    RichEdit1.SelStart := 0;
    flag := 0;
    curPos := 0;
    findNum := 0;
    downRadio.Checked := true;
    nr := chr(13);
    nrflag := 0;
    IsCase.Checked := true;

```

```
end;
```

该程序完整的代码可从中国水利水电出版社网站上下载。

5.15.4 执行程序

程序运行结果如图 5.46 所示，用户可以通过“打开文件”按钮加载一个文本文件到编辑器中，然后进行相应的编辑，或者直接输入文本字符进行编辑；通过“查找下一个”按钮搜索相应的字符串，后者通过“替换”按钮在编辑器中替换（所有的）被匹配的字符串。

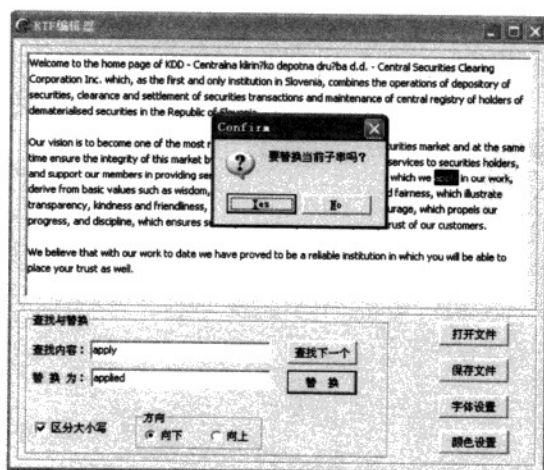


图 5.46 “RTF 编辑器”的运行结果

5.16 小结

本章主要介绍了 GUI 应用程序开发的方法和技巧，包括 VCL 组件的使用方法和技巧、单文档窗体和多文档窗体应用程序的设计和开发，以及 Delphi 应用程序菜单设计方法。通过对本章的学习，读者应掌握下列知识要点：

- VCL 组件的使用方法和技巧。
- 单文档窗体和多文档窗体应用程序的开发。
- Delphi 应用程序菜单设计方法。
- Delphi 中 GUI 应用程序开发的一般方法。

第6章 熟悉 Delphi 的 SQL 和 SQL Server 2000 的数据库管理

SQL Server 2000 是当今最流行的数据库管理系统之一，应用极为广泛。本章着重介绍了 SQL Server 2000 的使用方法，然后以此为平台详细介绍 SQL（结构查询语言）的基本语法，涉及的内容包括：

- Delphi 的数据库管理工具。
- SQL Server 2000 数据库和数据表的管理。
- SQL 的定义功能和操纵功能，包括表、视图、索引的创建、查询和删除的语法规则。
- SQL 的库函数和多表处理功能。

6.1 了解关系数据库

数据库系统的发展经历了数据的人工管理阶段、文件系统阶段乃至当今的大型数据库管理系统阶段。在这个发展过程中，曾一度形成了数据的层次模型、网状模型和关系模型之间的“三足鼎立”时期，但到目前为止却是关系模型“横霸天下”的时代。关系模型开创了数据库关系方法和关系数据库理论的研究，关系数据库理论已非常成熟和完善，绝大部分数据库系统都是基于关系模型的，其使用最为广泛。

数据库是一组数据的集合。关系数据库是使用二维表结构来存放数据的，表是关系数据库的基本组成部分。每一张数据表都是由行和列组成，行又称为记录，列称为字段（用于描述对象的属性）。不同的表可以通过它们之间的公共列相关。表中的数据具有下列特点：

- 没有完全相同的两行记录。
- 行的字段值是最小的数据单位，不能再分割了。
- 对表中的数据进行操作（插入、删除、修改等）时，必须满足数据的完整性和一致性。如表的主键值不能为空、不能引用不存在的记录及不能违反既定的约束规则等。

关系数据库系统有三部分，即数据库、数据库管理系统（DBMS）和数据库应用程序，如图 6.1 所示。

其中，数据库是数据表的集合，它以若干个文件的形式保存在计算机磁盘中。数据库管理系统（DBMS）是一种软件程序，用于描述、管理和维护数据库，对一个或者多个数据库进行统一管理。要访问数据库（中的数据），必须通过数据库管理系统。数据库应用程序是程序员为特定应用而开发的软件程序，它主要是通过对数据库进行存储访问实现提供既定的服务功能。作为 Delphi 程序员，如何编好一个高效而完善的数据库应用程序是主要任务。为此，必须首先了解 Delphi 数据库的管理工具及数据库的通用操作语言——SQL 语言。

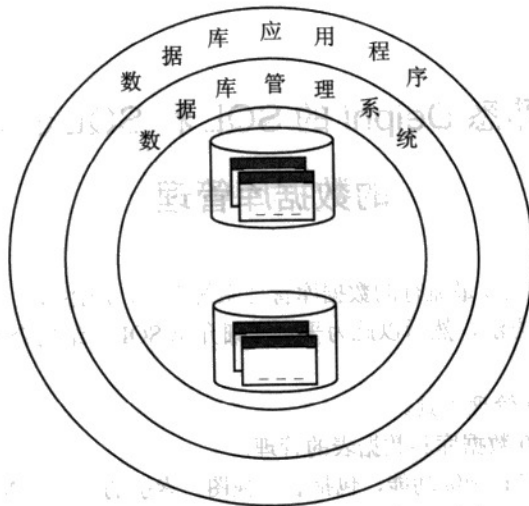



图 6.1 关系数据库系统的结构


6.2 熟悉 Delphi 的数据库管理工具

Delphi 2005 提供了 BDE Administrator 和 Database Explorer 两个管理工具，用于数据库的管理。

6.2.1 BDE Administrator

BDE (Borland Database Engine) 是 Borland 早期开发的数据库引擎 (Engine)，后经不断的改进和完善，现已提供强大的 API 调用函数库，几乎包含了所有的数据库驱动程序，可以对本 地及远程数据库进行连接和操作。例如，可以访问本地数据库 dBase、Access、FoxPro、Paradox 等；还可以访问远程数据库 InterBase、Oracle、MSSQL Server、Sybase、Informix 和 DB2 等。

BDE Administrator 是一种管理器 (简称 BDE 管理器)，主要用来集中管理数据库。它可以添加或修改一个数据库连接的名称，通常用于配置 BDE 别名和驱动程序。Delphi 2005 的启动菜单不包含 BDE 管理器的启动快捷命令。因此要启动 BDE 管理器，必须从 Windows 系统的控制面板中进行，方法是：选择菜单“开始”→“所有程序”→“控制面板”命令，然后在打开的控制面板中双击图标  即可打开 BDE 管理器，如图 6.2 所示。

如果打开控制面板后，看不到图标 ，则单击控制面板中的超链接“切换到经典视图”即可。

BDE 管理器有两个标签页：Databases 标签页和 Configuration 标签页，它们分别对应两种不同的功能。其中，Databases 标签页主要用于管理数据库别名，包括创建、删除和修改数据库别名，当在右边的框中选择某一个数据库别名时则在左边框中显示该别名的配置细节；Configuration 标签页则用于配置连接到各种数据库使用的参数，如使用服务器名等。

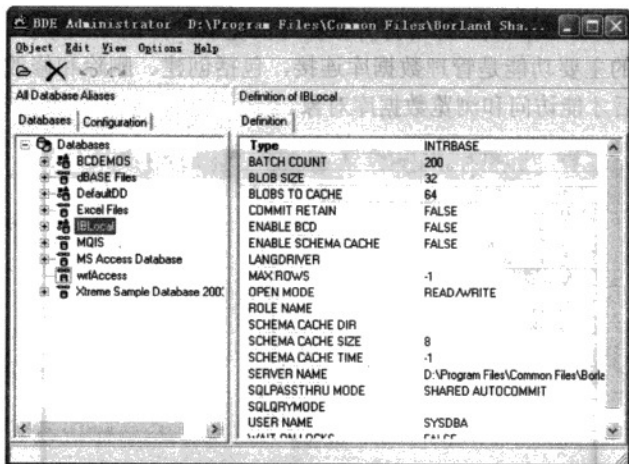


图 6.2 BDE 管理器

在窗口的左边选择 Databases 时，可以选择数据库的名称。如果某个数据库被选择，则将在右边的方框中显示到目前为止的数据库及其修改属性。

增加数据库别名时，可以选择 BDE 管理器的菜单 Object→New 命令，这时会弹出用于设置数据库种类的对话框，如图 6.3 所示。

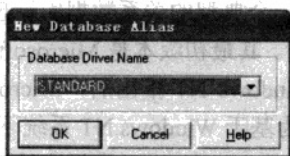


图 6.3 New Database Alias 对话框

设置完毕后单击 OK 按钮，这时会在左边的方框中出现一个新的数据库别名，同时它处于修改状态，用户可根据需要把它改为另外的名称。当数据库别名设置完成后，可在右边的方框中进一步设置数据库的相关属性，如数据库存储路径等。当所有的属性都设置完毕以后，选择菜单 Object→Apply 命令来保存设置结果，这时数据库别名的增加操作完毕。

对于数据库别名及其属性的修改，从上述的操作过程不难看出修改操作的一般方法。

如果数据库别名是用 ODBC 数据源管理器来配置的，则配置后的别名也都将在 BDE 管理器的 Databases 标签页中出现，同样可以对其进行管理和操作。

连接数据库时，可以单击数据库别名左边的“+”号，这时数据库的图标会变成绿色，表示数据库已经被连上。如果数据库设置用户名和密码时，在单击“+”号会弹出一个用于输入用户和密码的对话框，只要正确设置后单击 OK 按钮即可。

关闭数据库时，可以在数据库别名上右击，然后在弹出的快捷菜单中选择 Close 命令即可。

6.2.2 Database Explorer

Database Explorer 即数据库浏览器，是用于浏览和查询数据库的工具。通过它可以访问和查询如表、字段、触发器、存储过程和索引等数据库对象。通过选择“开始”→“所有程序”

→Delphi 2005→Database Explorer 即可打开数据库浏览器,如图 6.4 所示。

数据库浏览器的主要功能是管理数据库连接,包括创建、删除和修改数据库连接。只有成功地创建连接以后才能访问和浏览数据库对象。

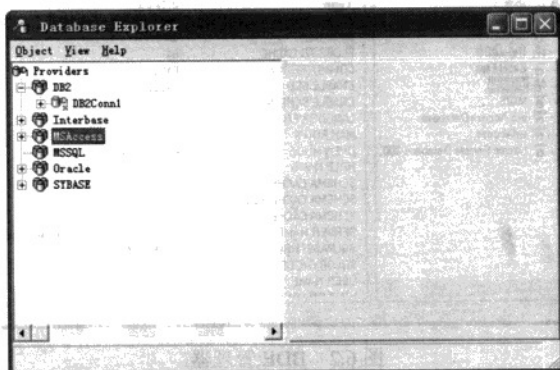


图 6.4 数据库浏览器

6.3 熟悉 Microsoft SQL Server 2000

Microsoft SQL Server 2000 是一个典型的关系数据库管理系统,是使用范围最为广泛和使用人数最多的一个数据库管理系统。其最初版本是 Microsoft、Sybase 和 Ashton-Tate 这三家公司合作开发的 OS/2 版本,但在 Windows NT 推出后,Microsoft 与 Sybase 在 SQL Server 的开发上就分道扬镳了,SQL Server 成为基于 Windows NT 系统下的一个关系数据库管理系统。SQL Server 2005 是 Microsoft 公司推出的 SQL Server 数据库管理系统的最新版本,但据说该版本是一个过渡版本,其各方面的性能有待于众人的实践验证。本书还是使用较为广泛应用的 Microsoft SQL Server 2000。为此,下面对 SQL Server 2000 的基本操作进行简要的介绍。

6.3.1 安装 SQL Server 2000

1. 安装 SQL Server 2000 的基本要求

SQL Server 2000 的常见版本有四种。

(1) 企业版 (Enterprise Edition)。企业版是最“完整”的一个版本,支持所有的 SQL Server 2000 特性,可作为企业 OLTP (联机事务处理) 及数据仓库系统等的产品数据库服务器。

(2) 标准版 (Standard Edition)。标准版一般用于小型的工作组或部门的数据库服务器,其同时承受用户的能力小于企业版。

(3) 个人版 (Personal Edition)。个人版一般用于个人用的数据库服务器,常用于单机系统或客户机。

(4) 开发者版 (Developer Edition)。开发者版用于程序员开发应用程序,这些程序需要 SQL Server 2000 作为数据存储设备。

安装 SQL Server 2000 所需的最低软件要求如表 6.1 所示。

表 6.1 安装 SQL Server 2000 的最低软件要求

版本	计算机	内存	硬盘空间
企业版 (Enterprise Edition)	Intel 兼容计算机, Pentium 166 MHz 以上	64 MB	完全安装 (Full) 180 MB
标准版 (Standard Edition)		32 MB	典型安装 (Typical) 170 MB
个人版 (Personal Edition)		32 MB	最小安装 (minimum) 65 MB
开发者版 (Developer Edition)		32 MB	只安装管理工具 (Client tools only) 90 MB Analysis Services: 50 MB English Query: 12 MB

SQL Server 2000 的四种版本对于不同的操作系统版本, 其可安装性是不相同的, 具体见表 6.2。

表 6.2 SQL Server 2000 的四种版本对操作系统版本的可安装性

操作系统版本	SQL Server 2000 版本			
	企业版	标准版	个人版	开发者版
Windows XP Professional	不	不	可以	可以
Windows 2000 Advanced Server	可以	可以	可以	可以
Windows 2000 Server	不	可以	可以	可以
Windows 2000 Professional	不	不	可以	可以
Windows 98	不	不	可以	不
Windows NT Server 4.0	不	可以	可以	可以

注: “不”表示不可以安装, “可以”表示可以安装。

2. 安装 SQL Server 2000 的基本步骤

经过以上检查并满足其要求后, 下面开始安装 SQL Server 2000。

(1) 如果从硬盘安装, 就直接双击 AUTORUN.EXE。当从光盘安装时, Autorun 屏幕就会自动出现。如果 Autorun 屏幕没有出现, 也可以直接在安装光盘的根目录下找到 AUTORUN.EXE 文件并双击, 则都会出现 SQL Server 2000 的安装界面。

(2) 在 SQL Server 2000 的安装界面中, 可以对四种版本的安装进行选择。在此, 选择个人版本, 在紧接着出现的画面中选择“安装 SQL Server 2000 组件”; 然后会出现组件安装界面, 选择数据库服务器, 出现 SQL Server 安装的欢迎界面, 单击“下一步”按钮, 则出现相应的安装向导界面。在该界面中选择“本地计算机”选项表示安装在本地计算机上, 选择“远程计算机”选项表示安装到远程计算机上; 选择“虚拟服务器”选项表示安装到虚拟计算机中, 个人版安装中不提供这个选项。

(3) 选择了“本地计算机”后, 单击“下一步”按钮, 出现如图 6.5 所示的“安装选择”对话框。

在此对话框中, 有三项可供选择。如果选中“高级选项”, 则会出现高级选项对话框, 其中有三个选项:

- 记录无值守.ISS 文件: 为自动安装创建一个初始化设置文件。

- 注册表重建：将一个损坏的安装修复。
- 维护故障转移群集的虚拟服务器：维护虚拟服务器。

(4) 在图 6.5 所示的对话框中选择“创建新的 SQL Server 实例，或安装‘客户工具端工具’”，则会弹出输入用户信息对话框，根据需要输入相应的信息。

(5) 单击“下一步”按钮，则会出现版权协议信息对话框。选择接受版权协议，则出现输入产品序列号对话框。根据购买产品时得到的序列号对该对话框进行设置，然后单击“下一步”按钮，则会出现安装定义对话框。

(6) 选择“服务器和客户端工具”选项，安装服务器和客户端工具，然后单击“下一步”按钮，则出现“实例名”设置对话框，如图 6.6 所示。

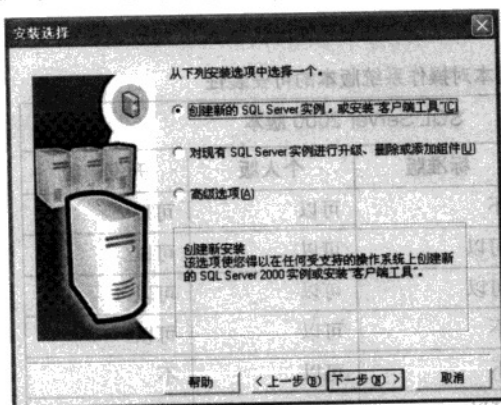


图 6.5 “安装选择”对话框

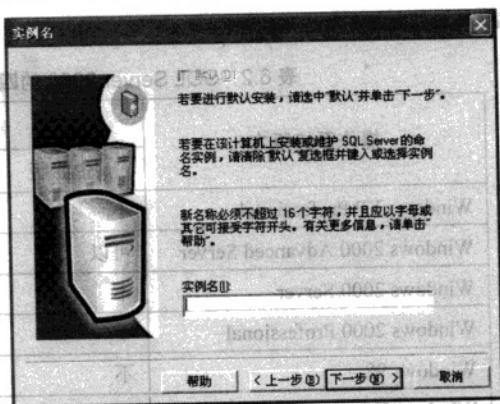


图 6.6 “实例名”设置对话框

在此对话框中，可以使用系统默认的实例名字（如果默认的实例名已被占用，则不能选择，如本例），也可以自定义实例名。

(7) 在此，把实例名设置为 MyInstance，单击“下一步”按钮，出现“安装类型”选择对话框，如图 6.7 所示。

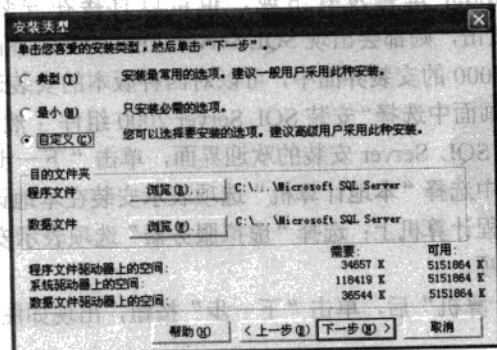


图 6.7 “安装类型”选择对话框

在此对话框中可以修改 SQL Server 2000 的安装类型、程序安装目录和数据文件的存放目录。其中“典型”安装不安装 SQL 代码示例文件和部分开发工具；选择“自定义”单选按钮，

则可以定制要安装的组件或完全安装 SQL Server 2000。

(8) 为了多了解一点 SQL Server 2000 的安装过程, 在安装类型选择对话框中选中“自定义”单选按钮, 单击“下一步”按钮, 出现“选择组件”对话框, 如图 6.8 所示。可根据需要选择所要安装的 SQL Server 组件。

(9) 选择完成后(见图 6.8), 单击“下一步”按钮, 出现“服务帐户”设置对话框, 如图 6.9 所示。

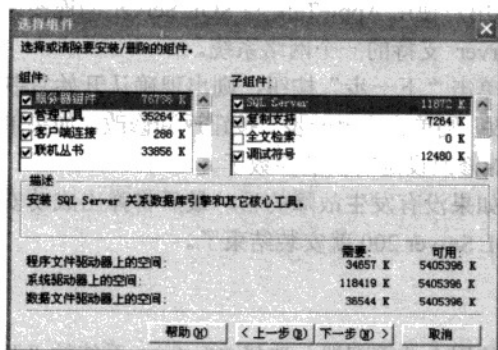


图 6.8 “选择组件”对话框

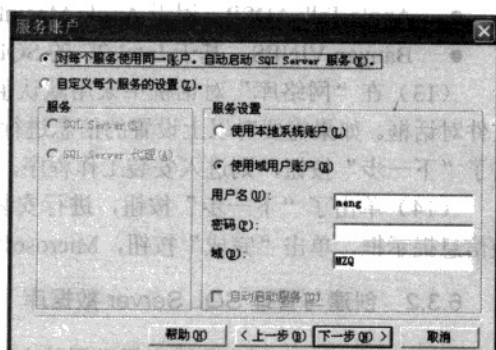


图 6.9 “服务帐户”设置对话框

其中, “服务设置”栏中, 系统默认的选择是“使用域用户帐户”。这时需要输入一个 NT 网络上的帐号作为 SQL Server 的启动帐号, 这个帐号必须已经由网域用户管理员建立, 位于 Administrators 区域组中, 且设定为密码永远有效。

(10) 由于笔者没有既定 NT 网络账号, 所以选择“使用本地系统帐户”(即使用 Windows 登录帐户和密码), 单击“下一步”按钮, 出现如图 6.10 所示的“身份验证模式”设置对话框。

(11) 选中“混合模式”(Windows 身份验证和 SQL Server 身份验证), 单击“下一步”按钮, 出现排序规则设置对话框。在此对话框中, 可以设置数据如何进行比较排序显示等处理操作的规则。

(12) 在排序规则设置对话框中采用默认值, 单击“下一步”按钮, 则出现“网络库”对话框, 如图 6.11 所示。

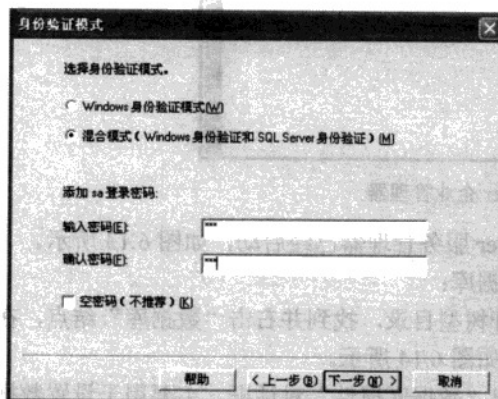


图 6.10 “身份验证模式”设置对话框

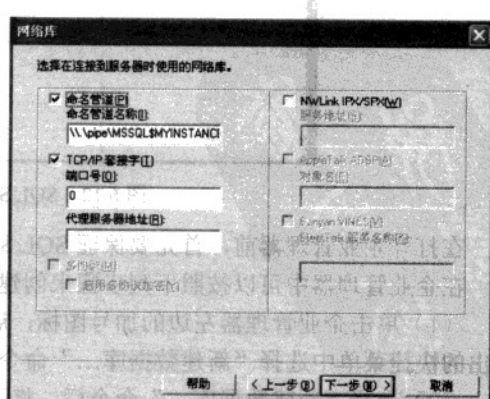


图 6.11 “网络库”对话框

各选项说明如下：

- 命名管道。SQL Server 默认的安装设置，用于在 NT 网络上允许本地或网络通信。
- TCP/IP 套接字。用于设置端口号（默认值为 0）和代理服务器地址（没有则留空白）。
- 多协议。多协议使用 Windows NT RPC（远程过程调用）结构来进行通信，它支持 NWLink IPX/SPX、TCP/IP 和命名管道协议。
- NWLink IPX/SPX。IPX/SPX 是指 Novell 网中的协议。
- Apple Talk ADSP。用于 Apple Macintosh 用户使用 AppleTalk 与 SQL Server 的连接。
- Banyan VINES。基于 Intel PC 的 SQL Server 支持的一个网络系统。

(13) 在“网络库”对话框中采用默认值，单击“下一步”按钮，则出现确认开始复制文件对话框。如果需要对以上设置的信息进行修改，则单击“上一步”按钮返回修改，如果单击了“下一步”按钮，则进入安装工作程序，不能修改设定的安装参数了。

(14) 单击了“下一步”按钮，进行安装。如果没有发生故障的话，最后将弹出成功安装信息提示框，单击“完成”按钮，Microsoft SQL Server 200 就安装结束了。

6.3.2 创建与管理 SQL Server 数据库

在 SQL Server 中创建数据库最常用的方法是使用企业管理器。选择 Windows 系统的“开始”→“所有程序”→Microsoft SQL Server→“企业管理器”命令，可以打开企业管理器，如图 6.12 所示。

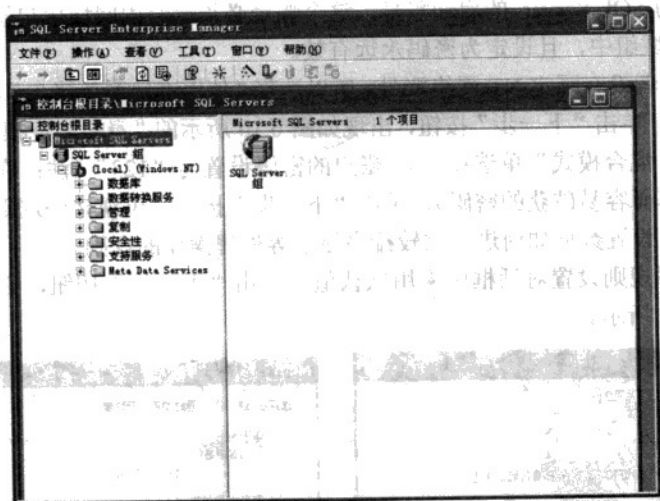


图 6.12 SQL Server 企业管理器

在打开企业管理器前，首先要保证 SQL Server 服务管理器已经启动，如图 6.13 所示。在企业管理器中可以按照下列步骤来创建数据库：

(1) 单击企业管理器左边的加号图标，展开树型目录，找到并右击“数据库”结点，在弹出的快捷菜单中选择“新建数据库...”命令，如图 6.14 所示。

(2) 选择“新建数据库...”命令后，将弹出“数据库属性”对话框，主要用于设置数据库的属性。

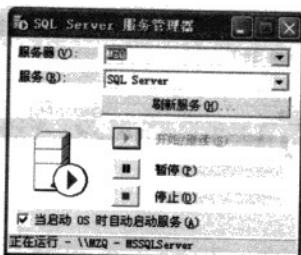


图 6.13 正在运行的服务管理器

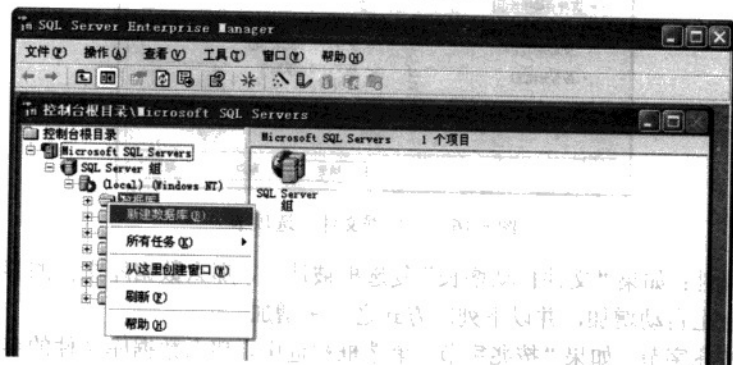


图 6.14 选择新建数据库命令

- 数据库名：在“名称”文本框中输入字符串作为待创建的数据库的名称，如输入 MyDatabase，如图 6.15 所示。

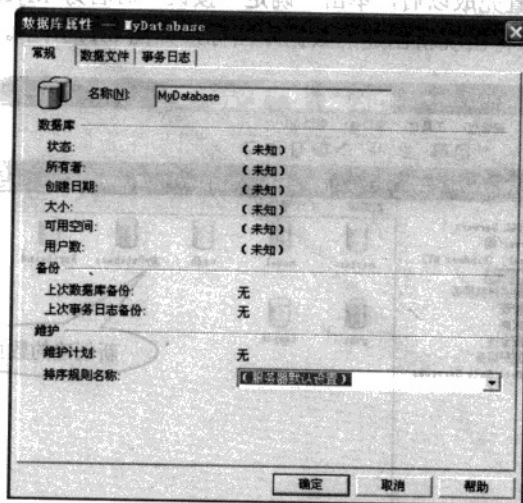


图 6.15 设置数据库名

- 数据库文件名和数据库初始容量：选择“数据文件”选项卡，可以在“数据库文件”列表框中设置待创建数据库的文件名、文件存放路径及数据库的初始容量，如图 6.16 所示。

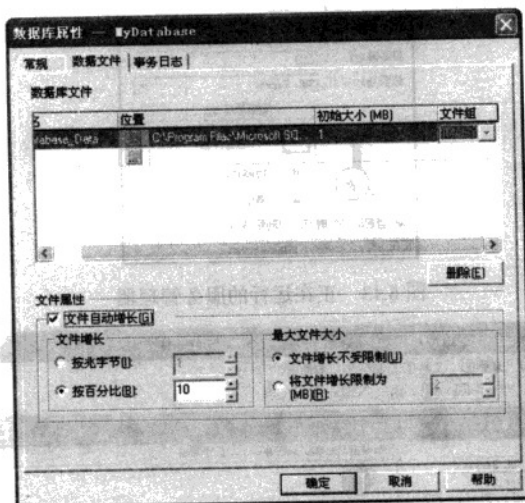


图 6.16 “数据文件”选项卡

- 文件属性：如果“文件自动增长”复选框被选中，那么数据库文件的容量会在初始值的基础上自动增加，并按以下列的方式之一来增加。
 - 按兆字节：如果“按兆字节”单选框被选中，那么数据库文件的容量将以兆字节为单位自动增加，随后的文本框中的数字表示容量增长的幅度。
 - 按百分比：如果“按百分比”单选框被选中，那么数据库文件的容量将以百分比为单位自动增加，其后文本框中数字表示增长的百分数。

(3) 数据库属性设置完成以后，单击“确定”按钮，命名为 MyDatabase 的数据库创建完毕。这时企业管理器中将出现新创建的数据库图标，如图 6.17 所示。

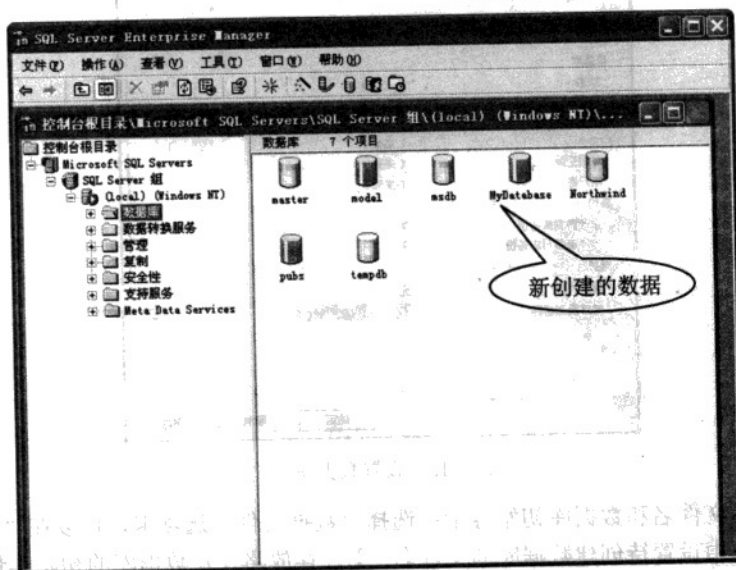


图 6.17 企业管理器（创建了数据库 MyDatabase）

除了上面介绍的数据库创建方法以外,还可以使用向导以及使用 Transact-SQL 语句的方法来创建数据库。但创建数据库的频率很小,掌握以上介绍的方法就足够了,所以不再赘述。

在企业管理器中可以通过双击数据库图标来修改数据库的各项属性,或者通过右击数据库图标来删除数据库等。

6.3.3 创建与管理 SQL Server 数据库表

数据库是表的集合,表是数据库中非常重要和常见的数据库对象。在 SQL Server 中,可以用企业管理器来创建和管理数据表。

在 SQL Server 企业管理器中,可以按照下列方法来创建数据库 MyDatabase 中名为 stu 的数据表:

(1) 启动企业管理器,展开其左边的树型目录,找到并右击 MyDatabase 结点,在弹出的快捷菜单中选择“新建”→“表”命令(见图 6.18),将打开表设计器。

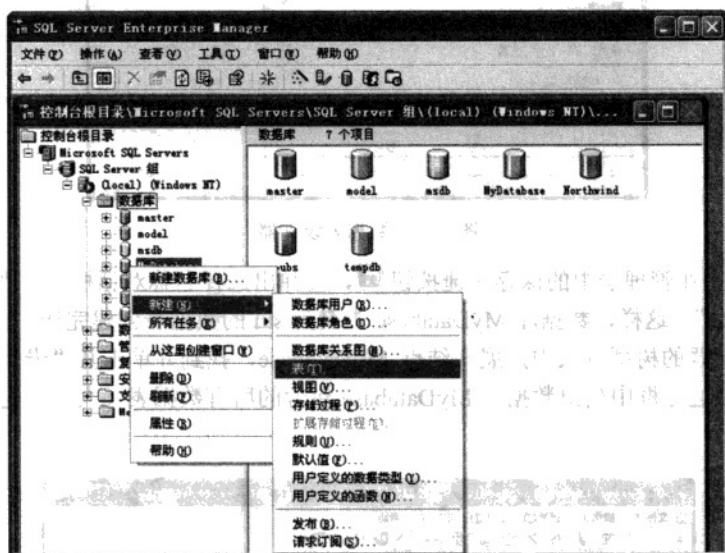


图 6.18 用弹出式菜单打开数据表设计器

(2) 在表设计器中,创建如表 6.3 所示的数据表。该表包括学号、姓名、性别、出生日期、籍贯、专业、成绩、备注等八个字段,并命名为 stu,设计结果如图 6.19 所示。

表 6.3 表 stu 的结构

名称	数据类型	长度	是否为空	说明
STU_ID	int	4	NOT NULL	学号
NAME	varchar	8	NOT NULL	姓名
GENDER	char	2		性别
BIRTHDAY	datetime	8		出生日期
NATIVE	char	20		籍贯

续表

名称	数据类型	长度	是否为空	说明
SPECIALITY	char	50		专业
GRADE	float	8		成绩
REMARK	varchar	500		备注

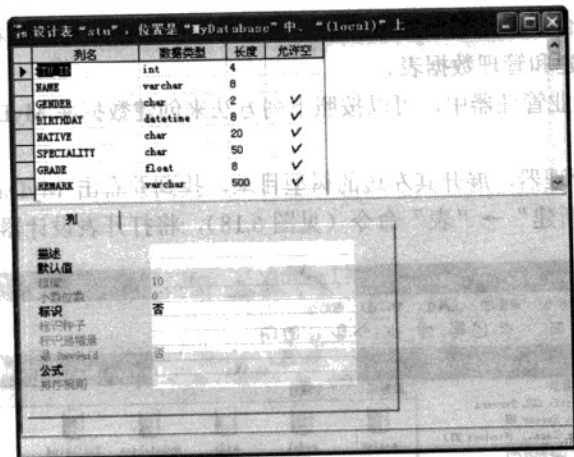



图 6.19 数据表设计器

(3) 单击企业管理器中的保存快捷按钮 ，会弹出选择名称对话框。在“输入表名”文本框中输入“stu”。这样，数据库 MyDatabase 中名为 stu 的数据表创建完毕。

在企业管理器的树型目录中，展开结点 MyDatabase，找到并单击其“表”子结点，则在企业管理器的右边框中列出数据库 MyDatabase 包含的所有数据表（大部分是系统数据表），如图 6.20 所示。

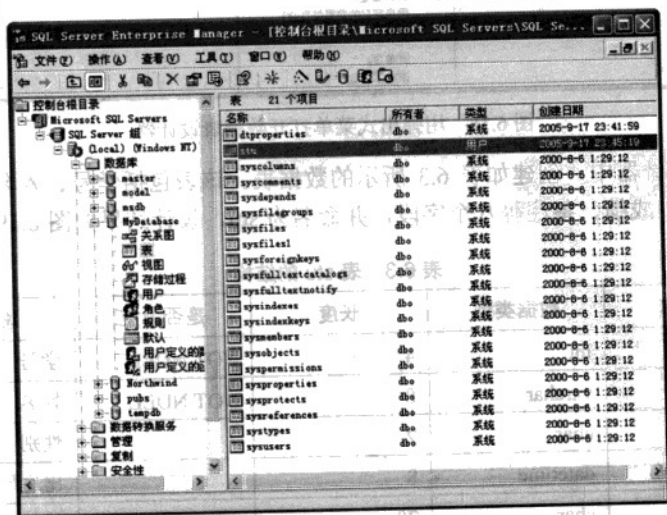


图 6.20 数据库 MyDatabase 所包含的数据表

如果要修改或删除某一个数据表，可以通过在右边的方框中右击相应数据表的图标，并在弹出的快捷菜单中分别选择“设计”和“删除”命令即可。


在本书以后部分，如果提到“数据库 MyDatabase”和“数据表 stu”就是指在本节创建的这个数据库 MyDatabase 和数据表 stu。

6.3.4 SQL Server 数据库的查询设计

数据库的查询可以利用查询分析器来完成。启动查询分析器的方法是：选择 Windows 系统的“开始”→“所有程序”→Microsoft SQL Server→“查询分析器”命令，可以打开查询分析器。不过首先打开的是连接 SQL Server 对话框。在此对话框中，选择“SQL Server 身份验证”单选框，并输入登录名为“sa”，密码为“123”（在 SQL Server 2000 安装时设置的），然后单击“确定”按钮即可进入“SQL 查询分析器”界面。在“SQL 查询分析器”可以用 SQL 语句对数据表和数据库进行操作。但是，SQL 语句对哪个数据库进行操作，这要通过选择快捷工具栏中的下拉列表框中相应的数据库名来完成。比如，查询数据库 MyDatabase 的 stu 表中的数据，则可以按照下列步骤来完成。

(1) 选择下拉列表框中的 MyDatabase 选项（表示对数据库 MyDatabase 的对象进行查询操作）。

(2) 在“查询”编辑框中输入查询语句，如“select * from stu”等。

(3) 单击工具栏上的执行查询按钮，即可在该编辑框的下部列出查询结果，如图 6.21 所示。

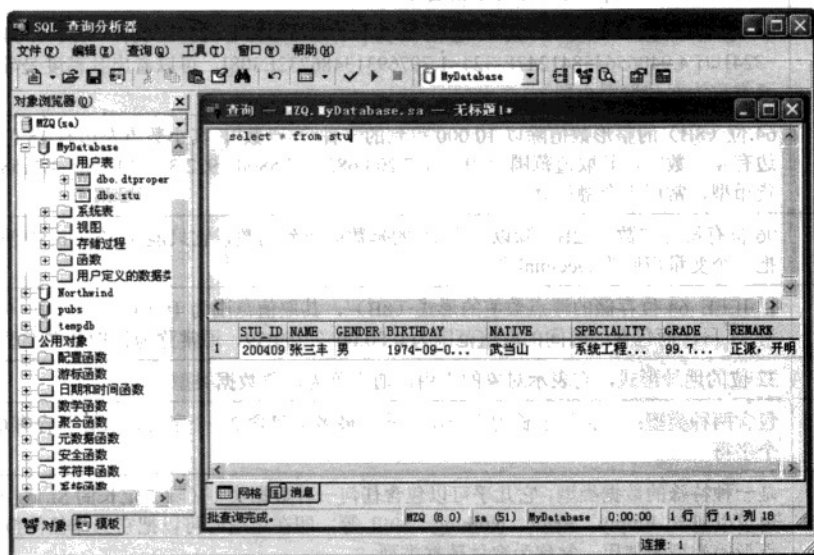


图 6.21 对数据库 MyDatabase 中的 stu 表进行查询

显然，在查询分析器中，除了可以查询数据以外，还可以进行创建、删除、修改数据表等操作，这将在下一节中结合 SQL 语言进一步介绍。

6.4 熟悉 SQL Server 的基本数据类型和开发工具

SQL 是英文 Structured Query Language 的缩写, 译为结构化查询语言。它于 1974 年由 Boyce 和 Chamberlin 提出, IBM 公司首先予以实现, 1984 年 10 月被 ANSI (美国国家标准化组织) 确定为数据库系统的工业标准。现在该语言已经成为数据库系统的通用语言, 特别是在 SQL Server 2000、Oracle、Foxpro 等关系数据库中得到了极为成功的应用。本章着重介绍在 SQL Server 2000 数据库系统环境下, 如何利用 SQL 这一利器来操纵和管理数据库。

数据类型反映数据的基本特征。在 SQL Server 中, 对表、视图等的定义离不开数据类型。为此, 首先对基本数据类型进行介绍, 然后再探讨 SQL 的有关功能。

SQL Server 提供了 12 种基本数据类型, 如表 6.4 所示。

表 6.4 基本数据类型

数据类型	说明
Byte	8 位无符号整数 (1B), 取值范围为 0~255, 通常称为字节型
Integer	16 位有符号整数 (2B), 取值范围为 -32 768~32 767, 通常称为整型
Long	32 位有符号整数 (4B), 取值范围为 -2 147 483 648~2 147 483 647, 通常称为长整型
Single	IEEE 32 位存储的浮点数, 它的取值范围为 $[-3.402823E38, -1.401298E-45] \cup [1.401298E-45, 3.402823E38]$, 通常称为单精度浮点型
Double	64 位浮点数 (8B), 其取值范围为 $[-1.79769313486231E308, -4.94065645841247E-324] \cup [4.94065645841247E-324, 1.79769313486232E308]$, 可以看出其精度是非常高的, 通常称为双精度浮点型
Currency	64 位 (8B) 的整形数值除以 10 000 得到的一种“浮点数”, 其小数点左边有 15 位数字, 右边有 4 位数字, 其取值范围为 -922 337 203 685 477.5808~922 337 203 685 477.5807, 称为货币型, 常用于金融计算
Decimal	96 位有符号整数 (12B) 除以一个 10 的幂数而得到的数, 它只能在 Variant 中使用, 不能把一个变量声明为 Decimal 类型
Date	以 IEEE 64 位存储的浮点数值的形式 (8B), 其取值范围为 100 年 1 月 1 日~9999 年 12 月 31 日, 而包含的时间的取值范围为 0:00:00~23:59:59, 通常称为日期数据类型
Object	32 位的地址形式, 它表示对象的引用, 通常称为对象数据类型
String	包含两种类型: 变长与定长的字符串。前者最多可包含 2^{31} 个字符, 后者可含有最多为 2^{16} 个字符
Variant	是一种特殊的数据类型, 它几乎可以包含任何一种数据类型 (除了定长的 String 类型外), 也可以包含 Empty、Error、Nothing、Null 等, 即在运用时可以把它当作任何整型或者浮点型等类型来使用。常称为变体数据类型
Boolean	以 16 位存储的数值形式, 其取值为 true 或 false

在 Microsoft SQL Server 中可以利用查询分析器来作为 SQL 语言的编程环境和开发工具。

6.5 熟悉 SQL 的定义功能

6.5.1 创建表

表是数据存储的基本单位。当需要存放数据时，首先要创建一个表。创建一个新表的语法如下：

```
CREATE TABLE 表名(列名 1      数据类型      [列约束条件][默认值],
                  列名 2      数据类型      [列约束条件][默认值],
                  ...          ...          ...
                  列名 n      数据类型      [列约束条件][默认值]
);
```

其中，列名是用户根据应用开发的约定来设定的；数据类型就是上面介绍的几种数据类型之一；[列约束条件]是用户设定的约束条件，如能否为空（NULL）、取值范围等；[默认值]是指用户没有输入数据之前，系统默认的值。

例如，如果要在数据库 MyDatabase 中创建如表 6.5 所示的数据表（表名为 stu），可以用下列的 SQL 语句：

```
CREATE TABLE STU("STU_ID" INTEGER NOT NULL,
                  "NAME"      VARCHAR(6) NOT NULL,
                  "GENDER"    INTEGER,
                  "BIRTHDAY"   DATETIME,
                  "NATIVE"     VARCHAR (20),
                  "SPECIALITY" VARCHAR(50),
                  "GRADE"     REAL,
                  "REMARK"     VARCHAR(500));
```

表 6.5 待插入的数据

学号	姓名	性别	出生日期	籍贯	专业	成绩	备注
1	宋远桥	男	1954 年 12 月 8 日	武当山	计算机应用	98	憨厚，正直，稳重
2	张翠山	男	1961 年 8 月 1 日	武当山	计算机软件	90	极多情重义，儒侠，看重道德
3	张无忌	男	1974 年 12 月 1 日	武当山	计算机软件	82	淡泊名利，运气好，为人正派
4	周芷若	女	1978 年 5 月 6 日	峨嵋山	数学	78	谋深，为情变态
5	赵敏	女	1978 年 12 月 18 日	蒙古	系统工程	82	倔强刚烈，为人聪明，痴情

把上述这个语句复制到查询分析器中，然后单击执行查询按钮 ►，即可在数据库 MyDatabase 中生成数据表 stu，如图 6.22 所示。

执行了该语句后就可以在分析器的左边方框中的 MyDatabase 结点下查到 stu 子结点，如图 6.23 所示。在此图中，可以查看表的结构及每个字段的名称和数据类型等信息。



图 6.22 创建数据表 stu

6.5.2 创建表的索引

在大型数据库系统中，如何有效地提高检索速度，是一个很重要的问题。对于一个表来说，加快其检索速度的一个最常用的方法是在相应的列上增加一个索引。

根据多种需要，可以对一个表建立若干个索引，以提供多种快速检索途径。建立索引的语法格式为：

```
create [unique] index 索引名
```

```
on 表名(列名 1 [次序], 列名 2 [次序], ..., 列名 p [次序]);
```

该语法表明，索引建立在 p 个列之上；小括号内的列名顺序说明了索引对列名依赖的顺序；“次序”是可选项，其值为 asc（升序）或 desc（降序），如果不选择则默认为 asc（升序）；unique 表示每一索引值只对应惟一的数据记录。

例如，对 stu 表中的 grade 列（降序）和 birthday 列（升序）建立索引，索引名为 index1，则相应语句为：

```
create unique index index1
on stu(grade desc, birthday);
```

查询分析器的执行结果如图 6.24 所示。

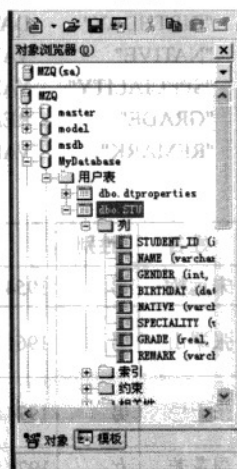


图 6.23 查看数据表 stu 对应的结点“dbo.STU”

6.5.3 创建视图

视图（View）可以看做一种表，有着与表一样的操作：用户可以对它进行查询和有限制的更新等。但是，视图并不是真正意义上的表（通常称为虚表），它是由一个或多个表（或视图）中的导出数据组成，是用查询来定义的，由查询获得数据，不分配给任何存储空间。存储在数据库中的视图实际上用于定义它的若干 SQL 语句，所以从本质上看，视图仅仅是一些 SQL

查询语句的集合。

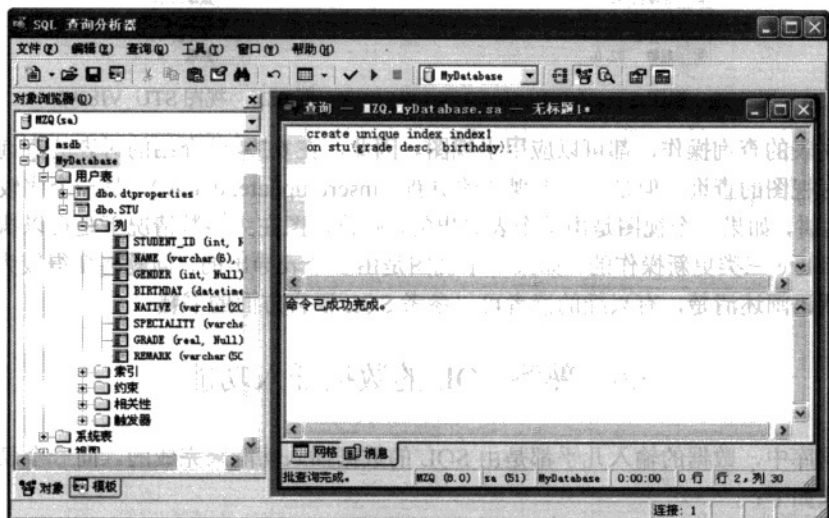


图 6.24 建立索引 index1

实际上,视图是为方便查看表中数据而提供的一种方法和途径。视图可以被形象地看做窗口。窗口的大小、位置是根据实际需要而定的,通过这个窗口用户可以方便地查看想要知道的数据,或进行其他的操作。

创建视图的 SQL 语句格式为:

```
create view 视图名[(列名 j1, 列名 j2, ..., 列名 jp)]
as select 列名 1, 列名 2, ..., 列名 p
from 表名
[其他]
```

其中,如果(列名 j₁, 列名 j₂, ..., 列名 j_p)被省略掉,则建立起来的视图的列名与列名 1, 列名 2, ..., 列名 p 一样。

例如,要建立由姓名和成绩组成的视图,可以用下列语句:

```
create view STU_VIEW
as select name, grade
from stu
```

当执行该语句以后,就可以像对表操作一样,查询其“包含”的数据。例如,用下列语句查询视图 STU_VIEW 所包含的所有数据:

```
select * from STU_VIEW;
```

在定义视图时,还可以为视图创建新的列名(而不用默认表中的列名)。例如,当用下列语句创建视图 STU_VIEW2,名为 name、grade 的字段分别变成名为 m1 和 m2:

```
create view STU_VIEW2(m1,m2)
as select name, grade
from stu
```

可用“select * from 视图名;”语句比较视图 STU_VIEW 和 STU_VIEW2,结果分别如图 6.25 和图 6.26 所示。

	NAME	GRADE
1	宋远桥	98.0
2	张翠山	90.0
3	张无忌	82.0
4	周芷若	78.0
5	赵敏	82.0

图 6.25 视图 STU_VIEW

	m1	m2
1	宋远桥	98.0
2	张翠山	90.0
3	张无忌	82.0
4	周芷若	78.0
5	赵敏	82.0

图 6.26 视图 STU_VIEW2

所有对表的查询操作，都可以应用于视图。因此，在 6.3.4 节介绍的对表的查询，实际上也是介绍对视图的查询。但是，对于视图的更新 (insert, update, delete)，是一个比较复杂的问题。一般地讲，如果一个视图是由单个表导出的，则该视图在大多数情况下是可以执行 insert、update 和 delete 三类更新操作的。如果一个视图是由多个表导出的，问题将变得较为复杂，难以用三言两语阐述清楚，有兴趣的读者可以参考 SQL 语言方面的书籍。

6.6 熟悉 SQL 的数据插入功能

在数据库中，数据的输入几乎都是由 SQL 的数据插入功能来完成的。向一个表插入数据的语法规则如下：

```
insert into 表名(列名 1, 列名 2, ..., 列名 m)
```

```
values(值 1, 值 2, ..., 值 m);
```

其中，值 1, 值 2, ..., 值 m 分别与列名 1, 列名 2, ..., 列名 m 对应，并且相应值与对应的列值类型相同。如果列名 1, 列名 2, ..., 列名 m 是被插入表的所有列，则“列名 1, 列名 2, ..., 列名 m”可以省略。

例如，对于表 6.5 中的数据可用下列语句把它们插入数据库 MyDatabase 的 stu 表中：

```
insert into stu values(1, '宋远桥', 1, '12/08/1954', '武当山', '计算机应用', 98, '憨厚', '正直', '稳重');
```

```
insert into stu values(2, '张翠山', 1, '08/01/1961', '武当山', '计算机软件', 90, '极多情重义', '儒侠', '看重道德');
```

```
insert into stu values(3, '张无忌', 1, '12/01/1974', '武当山', '计算机软件', 82, '淡泊名利', '运气好', '为人正派');
```

```
insert into stu values(4, '周芷若', 0, '05/06/1978', '峨眉山', '数学', 78, '谋深', '为情变态');
```

```
insert into stu values(5, '赵敏', 0, '12/18/1978', '蒙古', '系统工程', 82, '倔强刚烈', '为人聪明', '痴情');
```

当把上述的 SQL 语句复制到查询分析器中并单击执行查询按钮，则可完成 5 条数据的插入，如图 6.27 所示。

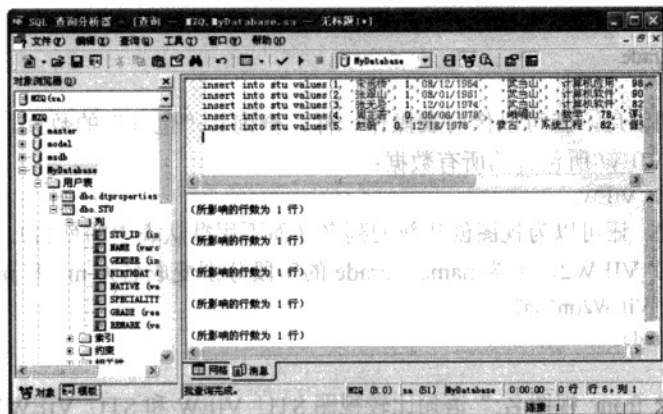


图 6.27 向 stu 表中插入 5 条数据

如果想向表中插入一条数据的部分信息,可用“表名后跟列名”的方法来插入。例如,下面的语句可用于向 stu 表中插入姓名为“殷离”,成绩为“80”的记录:

```
insert into stu(stu_id, name, grade) values(6,'殷离',80);
```

6.7 熟悉 SQL 的查询功能

数据库的查询是数据库应用的一个非常重要的功能,上面已经运用到了基本的查询语句。本节将系统地介绍 SQL 的查询功能。

6.7.1 基本查询语句

其格式为:

```
select 列名 1, 列名 2, ..., 列名 m from 表名 (或视图名);
```

该语句表示从被查询表 (或视图名) 中列出列名 1, 列名 2, ..., 列名 m 所包含的数据。列名 1, 列名 2, ..., 列名 m (列名之间用逗号隔开) 是该被查询表 (或视图名) 的所有列, 则可以以 “*” 代替 “列名 1, 列名 2, ..., 列名 m”, 即上面语句的格式变成:

```
select * from 表名 (或视图名);
```

它表示从被查询表 (或视图名) 中列出所有的数据, 如图 6.28 所示。

	STU_ID	NAME	GENDER	BIRTHDAY	NATIVE	SPECIALITY	GRADE	REMARK
1	1	宋远桥	1	1954-08-12 00:00:00.000	武当山	计算机应用	98.0	憨厚, 正直, 稳重
2	2	张翠山	1	1961-08-01 00:00:00.000	武当山	计算机软件	90.0	侠多情意重, 儒侠, 看重道德
3	3	张无忌	1	1974-12-01 00:00:00.000	武当山	计算机软件	82.0	淡泊名利, 运气好, 为人正直
4	4	周芷若	0	1978-05-06 00:00:00.000	峨眉山	数学	78.0	谋深, 为情变态
5	5	赵敏	0	1978-12-18 00:00:00.000	蒙古	系统工程	82.0	倔强刚烈, 为人聪明, 痴情
6	6	殷离	NULL	NULL	NULL	NULL	80.0	NULL

图 6.28 查询数据表 stu 的所有数据

如果只想查询表 stu 中所有人的名字 (name) 及成绩 (grade), 则可用下列语句:

```
select name, grade from stu;
```

其结果如图 6.29 所示。

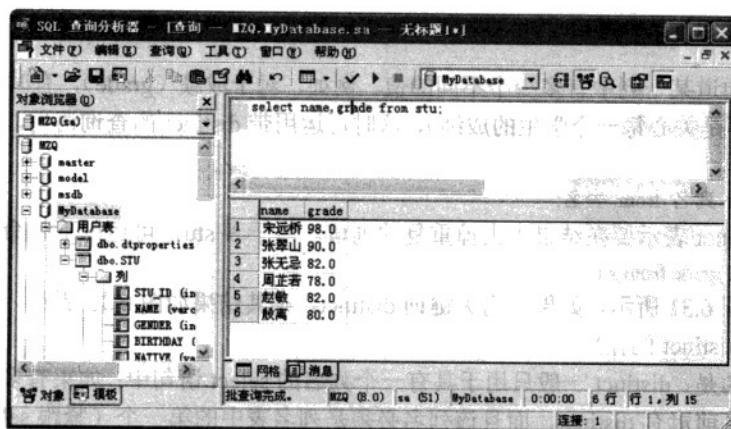


图 6.29 查询表 stu 中的部分数据

6.7.2 带 where 的条件查询

实际上,更多时候是根据一定的条件进行查询,即查询满足一定条件的部分数据(而不是表中的所有数据)查询。这时 where 子句将发挥作用,其一般语法格式如下:

```
select 列名 1, 列名 2, ..., 列名 m
```

```
from 表名 (或视图名)
```

```
where 条件表达式;
```

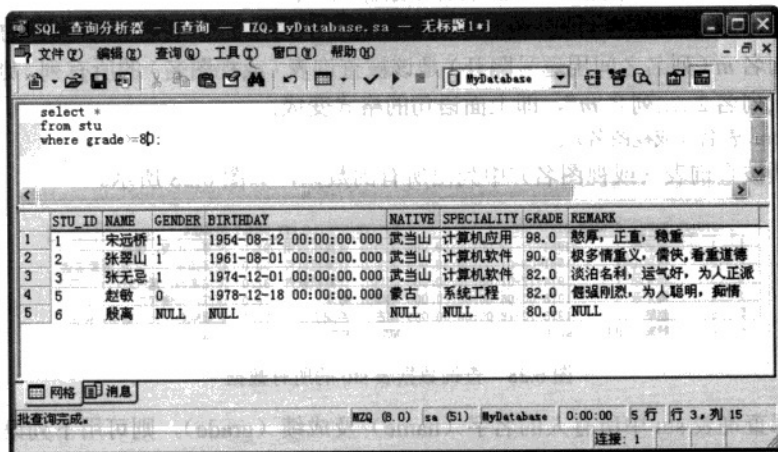
例如,在 stu 表中查找成绩 (grade) 大于或者等于 80 分的人,则用下列语句:

```
select *
```

```
from stu
```

```
where grade >= 80;
```

其结果如图 6.30 所示。其中,周芷若因为成绩仅为 78 分而没有显示在查询结果中。



STU_ID	NAME	GENDER	BIRTHDAY	NATIVE	SPECIALITY	GRADE	REMARK
1	宋远桥	1	1954-08-12 00:00:00.000	武当山	计算机应用	98.0	憨厚, 正直, 稳重
2	张翠山	1	1961-08-01 00:00:00.000	武当山	计算机软件	90.0	根多情重义, 爽快, 看重道义
3	张无忌	1	1974-12-01 00:00:00.000	武当山	计算机软件	82.0	淡泊名利, 运气好, 为人正派
4	赵敏	0	1978-12-18 00:00:00.000	蒙古	系统工程	82.0	倔强刚烈, 为人聪明, 痴情
5	殷离	NULL	NULL	NULL	NULL	80.0	NULL

图 6.30 成绩大于或等于 80 分的人 (查询结果)

6.7.3 带 distinct 的查询

有时希望知道某一列中有多少个不同的值。例如,对于成绩 (grade),有时希望了解它的分布情况(而不是关心每一个学生的成绩),这时可运用带 distinct 的查询语句,其一般语法格式如下:

```
select distinct 列名 from 表名;
```

其中, distinct 表示要在结果中去掉重复的列值。对于表 stu, 可运用下列命令进行查询:

```
select distinct grade from stu;
```

其结果如图 6.31 所示,如果不用关键词 distinct, 则其结果如图 6.32 所示。从这两个表中完全可以看出 distinct 的作用。

需要指出的是, distinct 一般只用于具有一个列名的 select 语句中;如果有多个列名,则只能有一个列名之前冠有 distinct, 而且该列名必须是列名表中的第一个(紧跟 select 之后);如果在多个列名之前冠有 distinct, 则该 select 语句是非法的。

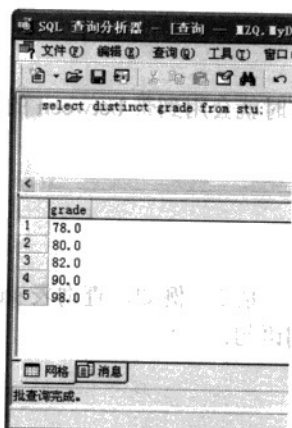


图 6.31 带 distinct 的查询结果

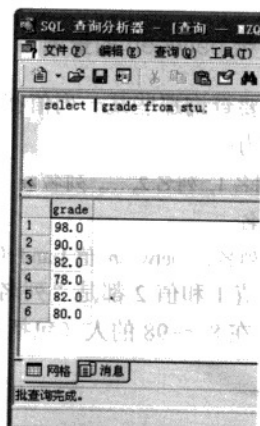


图 6.32 不带 distinct 的查询结果

6.7.4 有序查询

有序查询是指把查询结果按照某一列值的升序或降序进行显示，又称带 order 的查询，其语法格式为：

```
select 列名 1, 列名 2, ..., 列名 m
from 表名
order by 列名 i asc/desc;
```

其中，asc 和 desc 分别表示升序和降序。例如，对表 stu 按照成绩降序排列查询结果，可用如下命令：

```
select *
from stu
order by grade desc;
```

其结果如图 6.33 所示。如果需要按升序显示，只要把 desc 该为 asc。

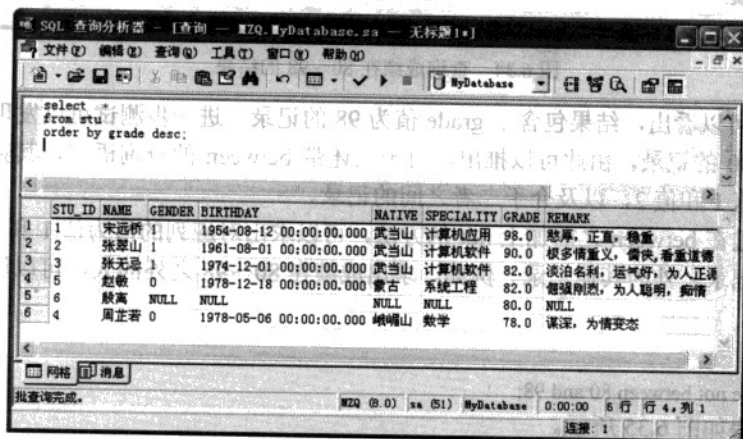


图 6.33 按成绩降序显示查询结果

6.7.5 带 between 的查询

有时需要查询那些某一列值在一定范围内的记录,这时就会用到带 between 的查询语句,其语法格式为:

```
select 列名 1, 列名 2, ..., 列名 m  
from 表名  
where 列名 i between 值 1 and 值 2;
```

其中,值 1 和值 2 都是“列名 *i*”的具体值,且值 1 < 值 2。例如,查询表 stu 中所有成绩 (grade) 在 80~98 的人 (包括 80 和 98),则可用下列语句:

```
select *  
from stu  
where grade between 80 and 98;  
其输出结果如图 6.34 所示。
```

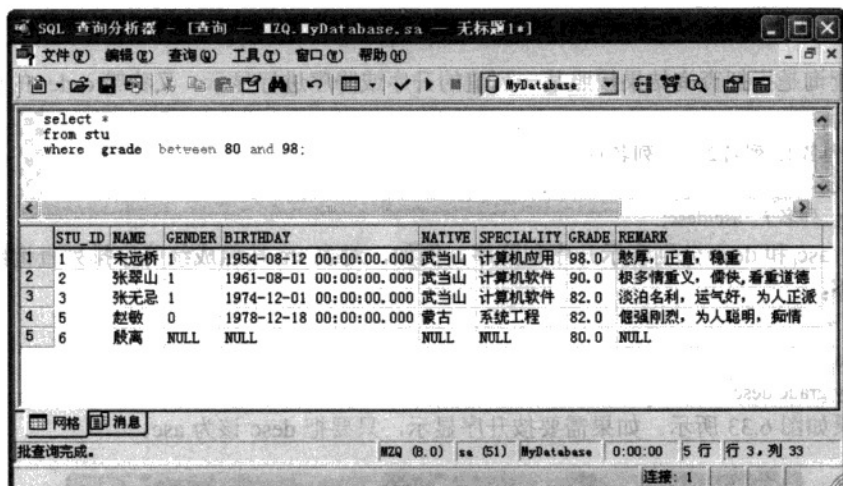


图 6.34 查询成绩在 80~98 的结果

从图 6.34 可以看出,结果包含了 grade 值为 98 的记录。进一步测试可以发现,结果也包含 grade 值为 80 的记录。由此可以推出,对于上述带 between 的查询语句,其查询结果包含对应列的值为值 1 和值 2,以及介于二者之间的记录。

另外,通过在 between 之前加上 not 的方法,可以求出对应列的值为值 1 和值 2,以及介于二者之间的记录之外的其他记录。例如,求出成绩在 80~98 之外的人,可以用下列语句:

```
select *  
from stu  
where grade not between 80 and 98;  
其输出结果如图 6.35 所示。
```



图 6.35 查询学生成绩不在 80~98 之间的学生记录

6.7.6 带 in 的查询

in 与 between 有类似的功能, 都是查询满足某列值在一定范围内的记录。但其与 between 不同的是, in 后面必须跟枚举的列值表, 即把所有的列值都列出来, 而不能写为“值 1 and 值 2”的形式。它适合于那些类型不是数值型的情况, 其语法格式为:

```
select *
from stu
where 列名 i in (列值 1, 列值 2, ..., 列值 p);
```

例如, 求出专业为“计算机应用”和“计算机软件”的学生, 则可用下列语句:

```
select *
from stu
where speciality in ('计算机应用','计算机软件');
```

查询结果如图 6.36 所示。

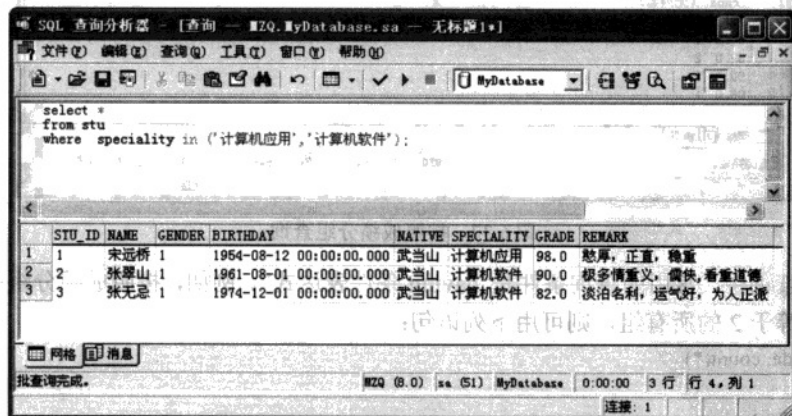


图 6.36 查询“计算机应用”和“计算机软件”专业的学生记录

实际上, “列名 i in (列值 1, 列值 2, ..., 列值 p)”等价于“列名 i = 列值 1 or 列名 i = 列值 2 or ... or 列名 i = 列值 p”。因此, 上例的查询语句也等价于:

```
select *
```

from stu

where speciality = '计算机应用' or speciality = '计算机软件';

另外, 与 between 类似, 对应列的值不在 (列值 1, 列值 2, ..., 列值 p) 中时, 可通过在 in 之前加上 not 来实现。对此不再举例。

6.7.7 带 group 的查询

带 group 的查询就是通常所说的分组查询, 它将查询结果按某一字段 (列) 分组显示, 其语法格式如下:

```
select 列名列表 1 [, count(列名 2)]
```

```
from stu
```

```
group by 列名列表 1
```

```
[having 条件表达式]
```

即按照列名列表 1 中的所有列来确定分组。其中, count(*) 是 SQL 的库函数 (这将在 6.11 节介绍), 通常与 group 结合使用, 表示每一组中列名 2 的值的数量。例如, 如果按照成绩分组并统计每一组中学生的人数, 则可用下列语句:

```
select grade, count(stu_id)
```

```
from stu
```

```
group by grade;
```

执行后, 其结果如图 6.37 所示, 其中 78、80、90、98 分各一人, 82 分有两人。

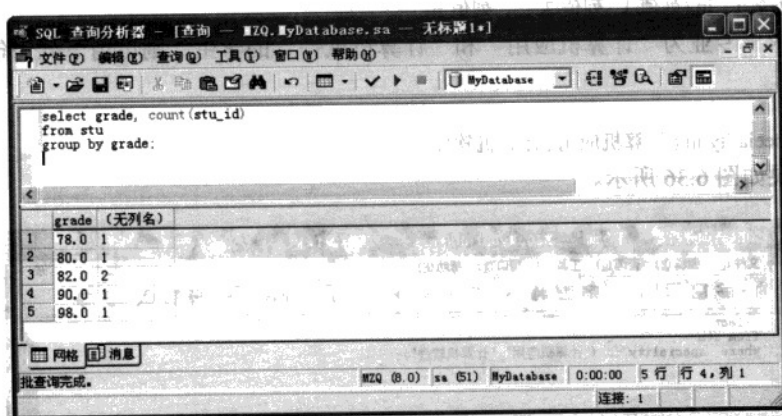


图 6.37 按照成绩分组查询

在分组查询中, 常用的查询条件是“having 条件表达式”。例如, 按照成绩分组并显示组中人数大于或等于 2 的所有组, 则可用下列语句:

```
select grade, count(*)
```

```
from stu
```

```
group by grade
```

```
having count(*) >= 2;
```

结果见图 6.38。如果把子句 “having count(*) >= 2” 改为子句 “having count(*) >= 1”, 则可以把所有的分组列出来, 这与没有子句 “having count(*) >= 1” 的效果是一样的。

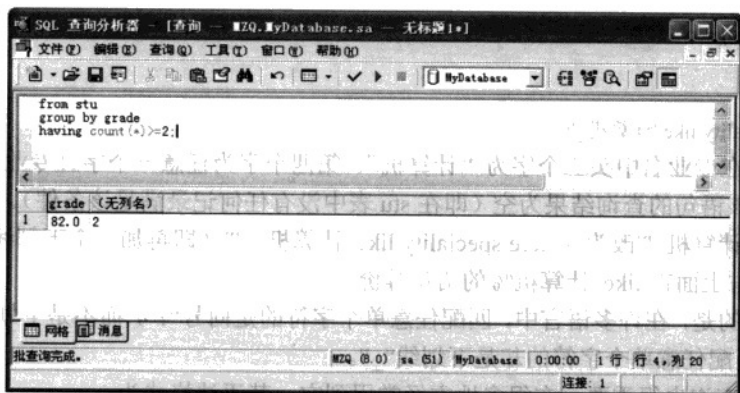


图 6.38 每组人数大于或等于 2 的分组查询

6.7.8 带 like 的查询和空值 null 的查询——实现模糊查询

在实际应用中，模糊查询用得很多。实际上，模糊查询在大多数情况下都是由带 like 的查询语句来实现的。带 like 查询的一般语法格式为：

```
select *
from stu
where 列名 i like 字符串常数;
```

其中，“列名 i”的类型必须是字符串类型；“字符串常数”有多种表达形式，不同的形式具有不同的功能。字符串“字符串常数”中的字符可以分为三种类型：

- “_”（下划线） 它可以与任意的单字符相匹配。
- “%”（百分号） 它可以与任意字符串（包括空串）相匹配。
- 除了字符“_”和“%”外，所有其他的字符都只能匹配自己。

例如，查询语句：

```
select *
from stu
where speciality like '计算机%';
```

表示欲查询专业名中头三个字为“计算机”的人，该语句执行结果如图 6.39 所示。

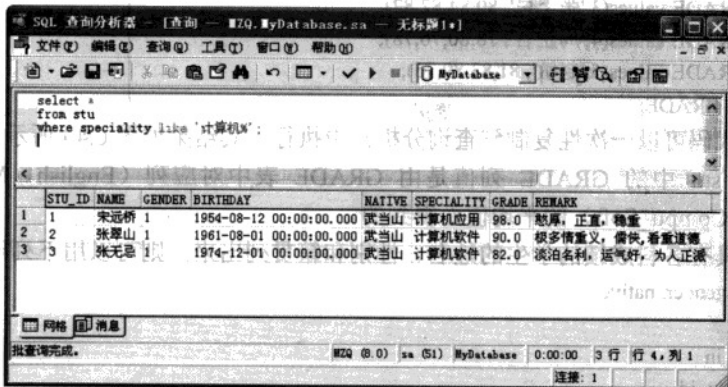


图 6.39 查询专业名中头三个字为“计算机”的结果

而查询语句:

```
select *  
from stu  
where speciality like '计算机_';
```

则表示要查询在专业名中头三个字为“计算机”、第四个字为任意一个字且专业名长度仅为 4 的人。显然,该语句的查询结果为空(即在 stu 表中没有任何记录满足该条件)。如果把“where speciality like '计算机_’”改为“where speciality like '计算机__’”(即再加一个下划线字符),则结果在 stu 表中与上面带 like '计算机%'的语句等价。

需要注意的是,在许多语言中,匹配任意单个字符的是问号“?”,而不是下划线“_”。但是,在 SQL 中,匹配任意单个字符的却是下划线“_”。

空值 null 查询也很重要,在很多地方经常用到它。其语法格式为:

```
select *  
from stu  
where 列名 is null;
```

上述语句中的最后一行不能写成“where 列名 = null;”。

6.8 熟悉 SQL 的嵌套查询

嵌套查询是由一个查询嵌套在另一个查询中构成的,被嵌套的查询称为子查询,包含子查询的查询称为父查询或者主查询。

为说明问题,先用下列语句构造名为 GRADE 的成绩表:

```
CREATE TABLE GRADE("STU_ID" INTEGER NOT NULL,  
    "NAME" VARCHAR(6) NOT NULL,  
    "English" REAL,  
    "Modern_Con" REAL,  
    "Compu_Appl" REAL,  
    "Compu_Soft" REAL);
```

} 创建表 GRADE

```
insert into GRADE values(1,'宋远桥',100,98,96,98);  
insert into GRADE values(2,'张翠山',95,93,87,85);  
insert into GRADE values(3,'张无忌',80,84,82,82);  
insert into GRADE values(4,'周芷若',78,80,76,78);  
insert into GRADE values(5,'赵敏',81,83,89,75);  
select * from GRADE;
```

} 添加数据并显示

以上两段代码可以一次性复制到查询分析器中执行,其结果如图 6.40 所示。

其中,表 stu 中的 GRADE 列值是由 GRADE 表中对应列(English、Modern_Con、Compu_Appl、Compu_Soft)的平均值而得到的。

如果要把具有各科成绩的学生的姓名、性别和籍贯列出来,则可以用下列嵌套语句:

```
select name, gender, native  
from stu  
where stu_id in  
    (select stu_id  
     from grade);
```

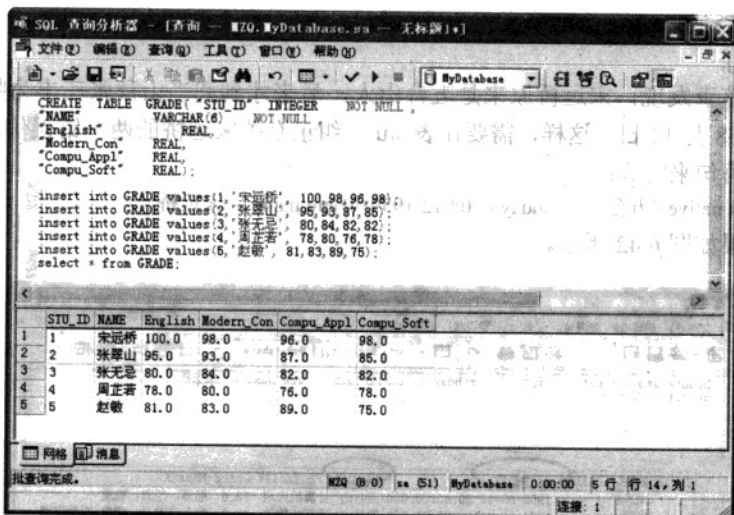


图 6.40 表 GRADE 的创建和数据显示

执行结果如图 6.41 所示。

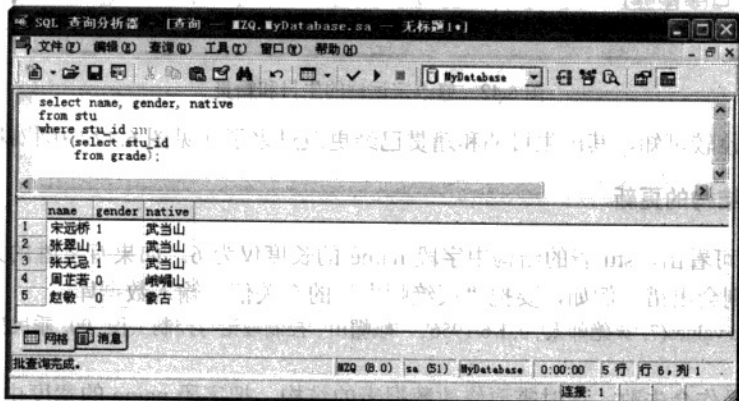


图 6.41 嵌套查询结果

在嵌套查询中，最常用的谓词是 in，此外，exist 和 not exist 等谓词也在嵌套查询中常用。但是，如果能够确认子查询返回的是单值时，还可以用“=”、“>”、“<”等比较查询。

6.9 熟悉 SQL 语言的更新功能

SQL 语言的更新功能主要包括数据的更新和表结构的更新。

6.9.1 数据的更新

数据更新的语法格式为：

```
update stu
```

set 列名 1 = 值 1, 列名 2 = 列值 2, ..., 列名 p = 列值 p

[where 子句];

例如, 经核查发现, 宋远桥原来是五台山人, 后来才到武当山拜张三丰为师的, 实际上是生于 1959 年 8 月 12 日。这样, 需要在表 stu 中纠正有关宋远桥的两个错误信息, 而这个操作可由下面的语句来完成:

```
update stu set native='五台山', birthday = '08/12/1959' where name = '宋远桥';
```

其输出结果如图 6.42 所示。

STU_ID	NAME	GENDER	BIRTHDAY	NATIVE	SPECIALITY	GRADE	REMARK
1	宋远桥	1	1959-08-12 00:00:00.000	五台山	计算机应用	98.0	憨厚, 正直, 稳重
2	张翠山	1	1961-08-01 00:00:00.000	武当山	计算机软件	90.0	极多情重义, 儒侠, 看重道义
3	张三丰	1	1974-12-01 00:00:00.000	武当山	计算机软件	82.0	淡泊名利, 运气好, 为人正派
4	周芷若	0	1978-05-06 00:00:00.000	峨眉山	数学	78.0	谨慎, 为情变态
5	赵敏	0	1978-12-18 00:00:00.000	蒙古	系统工程	82.0	倔强刚烈, 为人聪明, 痴情
6	殷离	NULL	NULL	NULL	NULL	80.0	NULL

图 6.42 修改宋远桥的生日和籍贯

与图 6.39 比较可知, 其出生日期和籍贯已经更改过来了 (见图 6.42 中圆圈标出的部分)。

6.9.2 表结构的更新

从图 6.42 可看出, stu 表的结构中字段 name 的长度仅为 6。如果有复姓的人名等长度超过 6 个字符, 则会出错。例如, 要把“灭绝师太”的有关信息输入数据库:

```
insert into stu-values(7, '灭绝师太', 0, '1/6/1956', '峨眉山', '控制理论与控制工程', 90, '乖戾狠辣, 刚愎自用, 残忍好杀');
```

则该插入操作会失败。这时需要修改数据表的结构, 把字段 name 的宽度由 6 改为 8 (或者更大) 才行。

表结构的更新分为以下几种 (但为安全起见, 对已经创建的表, 其结构修改受到很大的限制, 以下将予以适当的说明):

1. 列的增加

在表中增加一个列的语法格式为:

```
alter table 表名
```

```
add 列名 数据类型;
```

例如, 如果在 stu 表中欲增加一列——nationality (民族), 其长度为 20 个字符, 可由下列语句完成:

```
alter table stu
```

```
add nationality VARCHAR(20);
```

用 select 命令可以查看 stu 表结构的改变情况, 如图 6.43 所示。

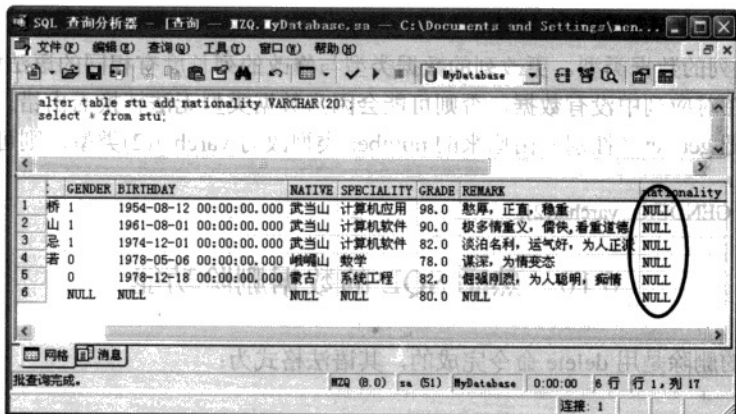


图 6.43 在 stu 表中增加了列 nationality

值得注意的是,当表为空时,可以增加约束为 NOT NULL 的列,否则(表不为空时)NOT NULL 是不允许的。另外,在表中增加的列都排在最后面。

2. 列的删除

在建表时,有时考虑欠佳,可能设置了多余的列,或者错误地增加了某一些列,这都需要把它们删除。列删除的语法格式如下:

```
alter table 表名 drop column 列名;
```

例如,要把上面增加的 nationality 列删除,可以用下列语句:

```
alter table stu drop column nationality;
```

3. 列的修改

列的修改包括增加列的长度、更改列的数据类型等。以下举例来对它们分别说明。

(1) 对列长度的修改。

对于表 stu,要把其 name 列(字段)的长度改为 8,则可用下列 SQL 语句:

```
alter table stu alter column name varchar(8);
```

在执行该语句后再执行下列语句则不会出错,结果如图 6.44 所示。

insert into stu values(7,'灭绝师太',0,'1/6/1956','峨嵋山','控制理论与控制工程',90,'乖戾狠辣,刚愎自用,残忍好杀');

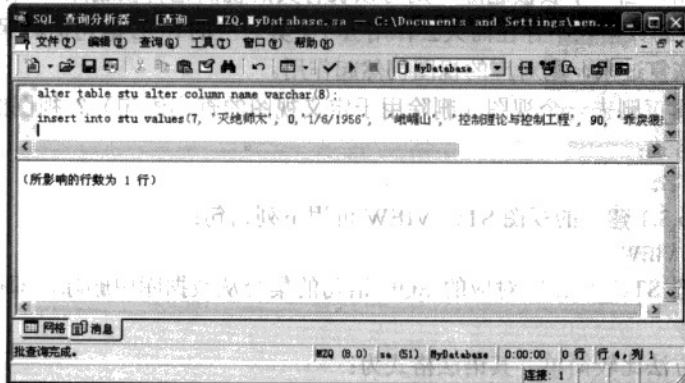


图 6.44 列长度的修改

列的长度只能改大，而不能改小。

(2) 更改列的数据类型。更改列的数据类型与修改的列长度有相同的语法格式。但其前提是，所有行在对应列中没有数据，否则可能会因为数据类型无法转换而出错。

例如，要把 gender (性别) 由原来的 number 类型改为 varchar(2) 类型，则可用下列语句：

```
alter table stu  
alter column GENDER varchar(2);
```

6.10 熟悉 SQL 的数据删除功能

表中数据的删除是用 delete 命令完成的，其语法格式为：

```
delete  
from 表名  
[where 条件表达式];
```

其中，“where 条件表达式”是可选的。如果不选择 where 子句，则默认把表中的所有数据全部删除；否则仅删除满足该条件的记录。例如，把成绩小于 90 的人从 stu 表中删除，可用以下命令：

```
delete  
from stu  
where grade < 90;
```

又如，把 stu 表中成绩在 90~100 的人删除，则可用下列命令：

```
delete  
from stu  
where grade between 90 and 100;
```

删除表的 SQL 语法格式为：

```
drop table 表名;
```

该语句将把表的定义、连同表中的所有数据以及与该表有关的所有索引、视图等全部删除，并释放相应的存储空间。

当在 SQL Server 中执行该语句时，SQL Server 没有给出任何的提示！如果是误操作，其结果之严重将是可想而知的。因此，当使用这一表删除语句时，应特别小心。

上面已经指出，当一个表被删除，则与该表有关的视图也将被删除。实际上，并不是删除，而是由于相应的数据表不存在而失去作用罢了。其对应的 SQL 语句仍然保存在数据库中，只要把相应的表恢复过来，则对应的视图仍然起作用。

那么，如何真正删去一个视图（删除用于定义视图的查询语句）？视图的删除与表的删除有类似的语法格式：

```
drop view 视图名;
```

例如，删除 6.3.1 建立的视图 STU_VIEW 可用下列语句：

```
drop view STU_VIEW;
```

这样就可以把 STU_VIEW 对应的 SQL 语句的集合从数据库中删除，即彻底清除了视图 STU_VIEW。

索引的删除方法比较简单，其语法格式为：

```
drop index 索引名;
```

例如，删除索引 index1 可以用下列语句：

```
drop index index1;
```

6.11 熟悉 SQL 的库函数

SQL Server 提供了许多非常有用的 SQL 库函数，这些函数增强了 SQL 的查询功能及查询的灵活性。以下对常用的几个函数分别予以介绍。

6.11.1 count 函数

count 函数具有两种形式：count(列名)和 count(*)。前者是计算对应列中非空值的个数，后者则是计算整个数据表中记录的个数。如果对应列中没有非空值，则 count(列名)和 count(*)的作用一样。另外，如果在“列名”前冠之以 distinct，则 count(distinct 列名)变成了计算对应列中不同非空值的个数。

对于图 6.45 所示的数据表 stu，可用下列语句求出 count(stu_id)的函数值：

```
select count(stu_id)
from stu;
```

其结果为 6，等于表中记录的条数。类似地，可以求出其他列的 count 函数值。

值得注意的是，count(*)在分组查询中并非计算整个数据表中记录的个数，而是计算每一组中的所有记录条数。

	STU_ID	NAME	GENDER	BIRTHDAY	NATIVE	SPECIALITY	GRADE	REMARK
1	1	宋远桥	1	1954-08-12 00:00:00.000	武当山	计算机应用	98.0	憨厚，正直，稳重
2	2	张翠山	1	1961-08-01 00:00:00.000	武当山	计算机软件	90.0	极多情重义，儒侠，看重道德
3	3	张无忌	1	1974-12-01 00:00:00.000	武当山	计算机软件	82.0	淡泊名利，运气好，为人正派
4	4	周芷若	0	1978-05-06 00:00:00.000	峨嵋山	数学	78.0	谋深，为情变态
5	5	赵敏	0	1978-12-18 00:00:00.000	蒙古	系统工程	82.0	聪明伶俐，为人聪明，痴情
6	6	殷离	NULL	NULL	NULL	NULL	80.0	NULL

图 6.45 表 stu 中的数据

表 6.6 中显示 count(*)的值及各列的 count(列名)值，从中可体会到 count(列名)和 count(*)的意义及其不同之处。

表 6.6 在 stu 表中 count(列名)和 count(*)的值

列名	函数值	列名	函数值
count(stu_id)	6	count(distinct stu_id)	6
count(name)	6	count(distinct name)	6
count(gender)	5	count(distinct gender)	2
count(birthday)	5	count(distinct birthday)	5
count(native)	5	count(distinct native)	3
count(speciality)	5	count(distinct speciality)	4
count(grade)	6	count(distinct grade)	5
count(remark)	5	count(distinct remark)	5
count(*)	6		

6.11.2 sum 函数、avg 函数、max 和 min 函数及 round 函数

sum 函数用于求取某一列值的总和, 此列的数据类型必须是数值型。例如, 要求出 stu 表中所有学生的总成绩, 则可用下列语句:

```
select sum(grade)
```

```
from stu;
```

其结果为 510.0。

avg 函数用于求某一列值的平均值, 此列的数据类型也必须是数值型。例如, 如果要求出 stu 表中学生成绩的平均值, 可用下列语句:

```
select avg(grade)
```

```
from stu;
```

其结果为 85.0。注意到, $85.0 = (98+90+82+78+82+80) \div 6$ 实际上, $\text{avg}(\text{列名}) = \text{sum}(\text{列名}) / \text{count}(\text{列名})$, 而不是 $\text{avg}(\text{列名}) = \text{sum}(\text{列名}) / \text{count}(*)$ 。

max 和 min 函数分别是求某一列值中的最大值和最小值。例如, 对于 stu 表, $\text{max}(\text{grade})$ 和 $\text{min}(\text{grade})$ 的值分别为 98 和 78。

round 函数具有形式 $\text{round}(x, n)$, 表示对 x 四舍五入, 保留 n 位小数位。例如, 执行下列语句:

```
select name, round(grade/3, 2)
```

```
from stu;
```

其结果如图 6.46 所示。

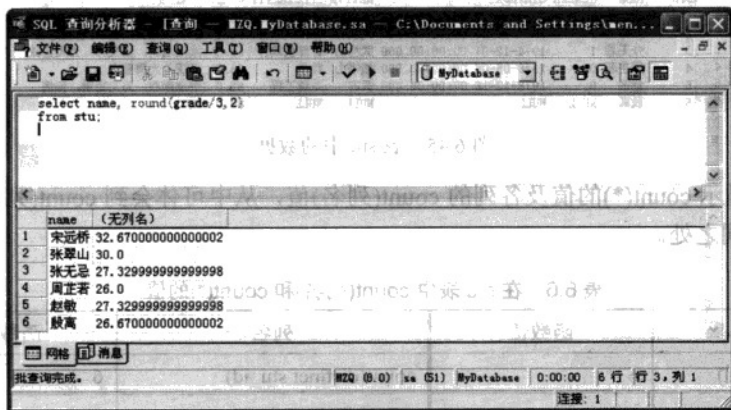


图 6.46 对 stu 表中个人成绩除以 3 并保留 2 位小数位

6.11.3 mod 函数、power 函数、floor 函数和 sign 函数

1. mod 函数

mod 函数的形式为 $\text{mod}(x, y)$, 其作用是求整数 x 除以整数 y 后的余数。例如, 下列语句将返回 2:

```
SELECT mod(5,3) as '5 除以 3 后的余数'
```

但此函数不受 Adaptive Server Enterprise 支持, 而 % 运算符在 Adaptive Server Enterprise 中

可作为模运算符使用。例如，下列语句也同样返回 2：

```
SELECT 5%3 as '5 除以 3 后的余数'
```

2. power 函数

power 函数的形式为 $\text{power}(x, n)$ ，表示计算 x 的 n 次方，即 $\text{power}(x, n) = x^n$ 。执行下列语句：

```
select name, grade, power(grade,2)
from stu;
```

其输出结果如图 6.47 所示。

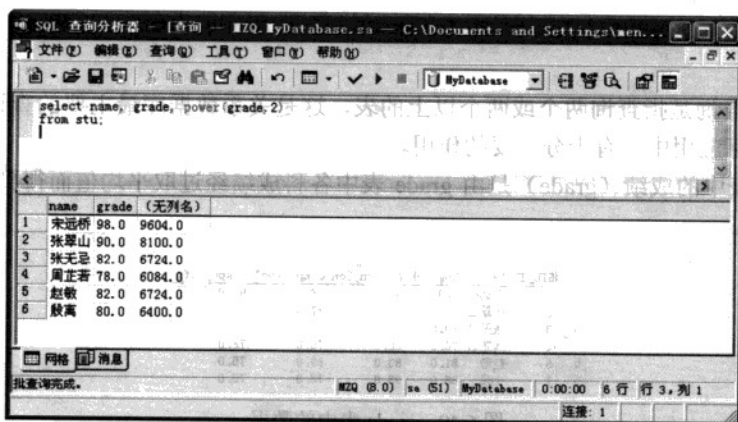


图 6.47 对 grade 列使用 power 函数

3. floor 函数

floor 函数的形式为 $\text{floor}(x)$ ，表示求取小于或者等于 x 的最大整数，即 $\text{floor}(x) = \lfloor x \rfloor$ 。

对于 stu 表，执行下列语句：

```
select name, grade, floor(grade/3)
from stu;
```

其输出结果如图 6.48 所示。

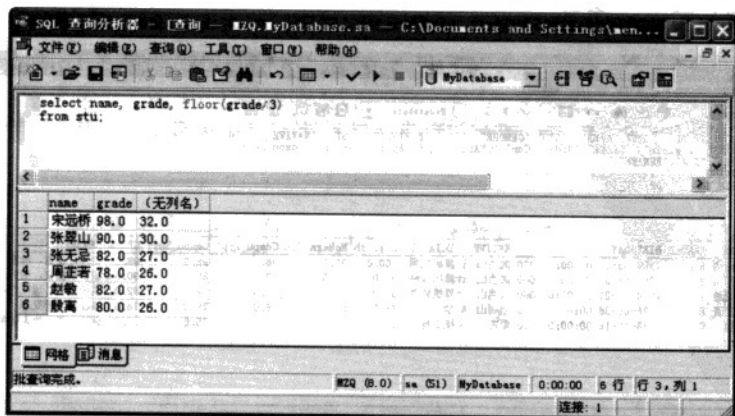


图 6.48 对 grade 列值除以 3 后使用 floor 函数

4. sign 函数

sign 函数的形式为 sign(x)，当 $x > 0$ 时，该函数值为 1；当 $x = 0$ 时，该函数值为 0；当 $x < 0$ 时，该函数值为 -1。即

$$\text{sign}(x) = \begin{cases} 1 & \text{当 } x > 0 \\ 0 & \text{当 } x = 0 \\ -1 & \text{当 } x < 0 \end{cases}$$

6.12 熟悉 SQL 的多表处理功能

多表连接查询是指查询两个或两个以上的表，这是关系数据库最主要的查询功能。多表连接查询在实际应用中具有十分重要的作用。

假设 stu 表中的成绩 (grade) 是由 grade 表中各科成绩经过取平均值而得到的。grade 表如图 6.49 所示。

STU_ID	NAME	English	Modern_Con	Compu_Appl	Compu_Soft
1	宋远桥	100.0	98.0	96.0	98.0
2	张翠山	95.0	93.0	87.0	85.0
3	张无忌	80.0	84.0	82.0	82.0
4	周芷若	78.0	80.0	76.0	78.0
5	赵敏	81.0	83.0	89.0	75.0
6	小昭	86.0	93.0	78.0	75.0

图 6.49 grade 表中的数据

如果要列出学生的基本信息及其各科成绩时，就会涉及两个表 (stu 表和 grade 表) 的查询，可用下列语句：

```
SELECT "STU"."STU_ID", "STU"."NAME", "STU"."GENDER", "STU"."BIRTHDAY",
"STU"."NATIVE", "STU"."SPECIALITY", "GRADE"."English", "GRADE"."Modern_Con",
"GRADE"."Compu_Appl", "GRADE"."Compu_Soft",
"STU"."GRADE", "STU"."REMARK"
FROM STU, GRADE
WHERE STU.STU_ID = GRADE.STU_ID;
```

其输出结果如图 6.50 所示。

STU_ID	NAME	GENDER	BIRTHDAY	NATIVE	SPECIALITY	English	Modern_Con	Compu_Appl	Compu_Soft	GRADE	REMARK
1	宋远桥	1	1964-08-12 00:00:00.000	武当山	计算机应用	100.0	98.0	96.0	98.0	98.0	聪明，正直，稳重
2	张翠山	1	1961-08-01 00:00:00.000	武当山	计算机软件	95.0	93.0	87.0	85.0	90.0	积极向上，为人正直
3	张无忌	1	1974-12-01 00:00:00.000	武当山	计算机软件	80.0	84.0	82.0	82.0	82.0	淡泊名利，运气好，为人正直
4	周芷若	0	1978-05-06 00:00:00.000	峨眉山	数学	78.0	80.0	76.0	78.0	78.0	调皮，为人正直
5	赵敏	0	1978-12-18 00:00:00.000	蒙古	系统工程	81.0	83.0	89.0	75.0	82.0	聪明，为人正直，为人聪明，病情

图 6.50 关于 STU_ID 的等值连接查询

其中, “STU.STU_ID = GRADE.STU_ID”称为连接条件, 表示被查询的表中满足该条件的记录已被列出。例如, 对于 grade 表中的 stu_id 为 7 的记录, 由于在 stu 表中没有 stu_id 为 7 的记录, 所以没有满足连接条件而未能列出。

对选作连接条件的列, 它们的数据类型最好是相同的, 至少是可比的。除了上述的比较符“=”外, 还可以是“<”、“>”、“<=”、“>=”等。当比较符为“=”时, 称为等值连接查询, 这是用得比较多的, 也是比较重要的一种多表查询。

如果某一列名在各个表中是惟一的(与所有的其他列名不同), 则可以把列名前的表名去掉。例如, 列名“English”在各表中是惟一的, 所以其前面的表名“GRADE”可以去掉, 而把“GRADE.English”写成“English”。

除此以外, 还有其他许多查询方法, 只要查询条件是合法的, 都可以产生相应的查询结果。

6.13 小结

本章着重介绍了 SQL Server 2000 的使用方法, 然后以此为平台详细介绍了 SQL (结构查询语言) 的基本语法。通过对本章的学习, 读者应掌握下列内容:

- 使用 Delphi 数据库的管理工具——BDE Administrator 和 Database Explorer。
- SQL Server 2000 数据库和数据表的管理, 包括安装、创建与维护数据库。
- SQL 的定义功能和操纵功能, 包括表、视图、索引的创建、查询和删除的语法规则。
- SQL 的库函数及其应用。
- SQL 的多表处理功能。

第 7 章 基于 BDE 的数据库应用开发

本章将介绍 Delphi BDE 数据库开发技术,首先介绍常用的 BDE 组件及相关数据库组件的属性和方法,然后介绍了数据库管理的基本操作,如数据浏览、查询设计、数据的插入、数据的删除、数据的修改等,最后用一个具体的例子对所学的内容进行了总结。本章涉及的内容要点包括:

- 常用的 BDE 组件及相关的数据库组件。
- 数据库连接的打开和关闭。
- 数据库的基本管理操作,包括数据浏览、查询设计、数据的插入、数据的删除及数据的修改等。
- BDE 数据库开发的基本方法。

7.1 关于 BDE

BDE (Borland Database Engine) 是 Borland 为了存取数据库而开发的数据库引擎。通过 BDE 可以非常方便地存取数据,使得对于各种不同的数据库格式可以轻松地运用 SQL 语句进行操作,而不必考虑各种数据的存放格式,从而使得 Delphi 对各种数据库的操作有了统一的接口。

BDE 本身支持面很广,它还可以连接到 ODBC。ODBC 是由微软开发的 Windows 系统的附属产品,它几乎可以连接到所有的数据库。因此,凡是 ODBC 支持的数据库,在理论上 BDE 也是可以支持的。但是,由于 ODBC 是“通”而不“精”,其数据存取速度并不理想,所以尽可能地不要采用“BDE+ODBC”的连接方式,这样会导致存取速度非常缓慢,并且数据存取也不稳定。

显然,与 ODBC 类似,BDE 是 Delphi 应用程序和数据库之间的一个接口,它同时可以在多个程序和多个数据库之间进行服务。当对其下达 SQL 命令时,它会自动解决不同数据库之间不同使用方式的限制,自动作出数据格式转换。BDE 在数据存取过程中的作用如图 7.1 所示。

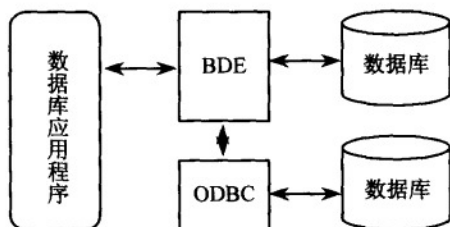


图 7.1 BDE 的作用

需要指出的是,Delphi 应用程序并不是一定要用 BDE 来连接数据库,还有许多其他连接

方式,这将在后续章节中介绍。

7.2 熟悉数据库的连接和断开

7.2.1 第一个BDE数据库应用程序

在本节中用 Delphi 2005 BDE 组件显示 SQL Server 数据库中表 stu(该表的创建方法见 6.5.1 节) 包含的数据,这相当于数据库应用程序的 Hello World 程序。

为创建数据库应用程序,首先要配置 ODBC 数据源。因此,以下先介绍创建名为“MyDSN”的 ODBC 数据源的方法,该 ODBC 数据源用于连接在 6.3.2 节创建的数据库——MyDatabase 数据库;然后再介绍一个简单的数据库应用程序。

1. 创建 ODBC 数据源

ODBC 数据源实际上是数据库的一个别名,通过这个别名应用程序就可以访问数据库。创建系统级别的数据源,可以通过下列步骤完成:

(1) 打开 ODBC 数据源配置对话框,方法是在 Windows 系统中选择“开始”→“控制面板”命令打开控制面板对话框(注:笔者用的是 Windows 操作系统),在控制面板中单击“性能和维护”超链接,然后在出现的性能和维护对话框中单击“管理工具”超链接,则出现管理工具对话框,最后在此对话框中双击“数据源 (ODBC)”图标即可打开 ODBC 数据源配置对话框。在对话框中选择 System DSN 标签页,表示要创建系统级的 ODBC 数据源。

(2) 在 ODBC 数据源管理器中单击 Add 按钮,将弹出创建数据源对话框,此对话框中列出了各种数据源的驱动程序。

(3) 在创建数据源对话框中选择 SQL Server 选项,然后单击“完成”按钮,将打开创建新数据源对话框。在此对话框中可对下面三个项目进行设置:

- Name 项。该项用于设置数据源的名称,在此输入“MyDSN”,即把待创建的数据源命名为“MyDSN”。
- Description 项。用于对创建的数据源进行文字说明,可自行决定是否设定。
- Server 项。该项是一个下拉列表框,当单击该列表框时它将列出本机所在局域网中所安装 SQL Server 的机器名。在此选择 local,表示用本地机作 SQL Server 数据库服务器。

(4) 设置完成后单击“下一步”按钮,将打开用户验证对话框。在此对话框中可以选择验证方式:一种是使用网络登录 ID 的 Windows 验证;另一种是使用用户登录 ID 和密码的 SQL Server 验证。在此选择后者,这时 Login ID 框和 Password 框被激活,分别输入 sa 和 123(注意,sa 和 123 是在 SQL Server 安装时设置的)。

(5) 设置完毕后,单击“下一步”按钮,将出现选择数据库对话框。默认连接的数据库是 master 数据库。如果想连接其他数据库,则可选中“Change the default database to:”复选框,这时其下面的下拉列表框被激活,单击该下拉列表框,可以选择 SQL Server 中的任意一个数据库。在此选择在前一章创建的数据库——MyDatabase。

(6) 选择了 MyDatabase 数据库以后(其他项采用默认值),单击“下一步”按钮。在弹出的对话框中选择默认设置,然后单击“完成”按钮,将弹出数据源设置信息对话框。

(7) 在数据源设置对话框中, 给出了有关前面几步所做设置的信息。如果设置正确, 那么当单击 Test Data Source 按钮时将弹出数据源创建成功的信息。

(8) 在数据源测试成功后, 单击 OK 按钮, 数据源 MyDSN 创建完毕。这时在 ODBC 管理器中将看到新生成的数据源 MyDSN, 如图 7.2 所示。

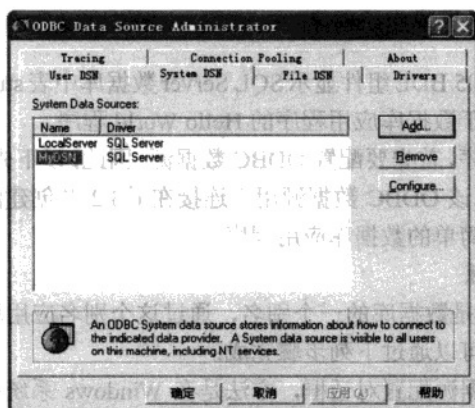


图 7.2 数据源 MyDSN 已被创建

在很多数据库应用程序中都会应用到 ODBC 数据源, 希望读者能够通过上面的例子举一反三, 熟练掌握数据源的创建方法。

2. 创建数据库应用程序

在 Delphi 2005 中创建一个应用程序, 不妨采用默认的工程文件名 project1.dpr。在本节中, 将利用已创建的数据源 MyDSN 来构造一个只有简单浏览功能的数据库应用程序。

(1) 在新创建的 VCL 工程 project1 的窗体中分别添加一个 Table 组件、一个 DataSource 组件和一个 DBGrid 组件, 并对 DBGrid 组件作适当的调整, 结果如图 7.3 所示。

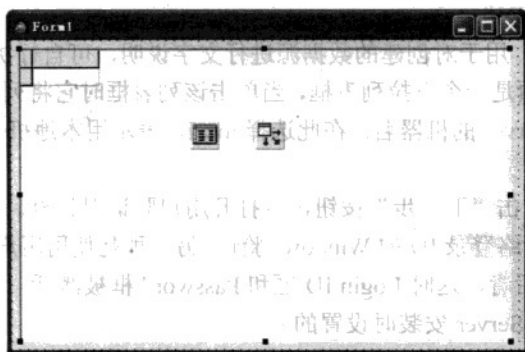


图 7.3 窗体组件设置

Table、DataSource 和 DBGrid 组件分别位于工具面板中的 BDE、Data Access 和 Data Controls 标签页上。

(2) 当把以上组件都放于窗体中并作适当调整后, 分别对各组件的属性进行设置, 结果如表 7.1 所示。

表 7.1 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Table	Table1	DatabaseName	MyDSN
		TableName	dbo.STU
		Active	true
DataSource	DataSource1	DataSet	Table1
DBGrid	DBGrid1	DataSource	DataSource1
Form	Form1	Caption	简单数据库应用程序

其中, MyDSN 为上面创建的 ODBC 数据源。当设置 TableName 属性值时将弹出数据库登录对话框。在两个文本框中分别输入 sa 和 123 即可(sa/123 是在安装 SQL Server 时设置的)。

最后设置的是组件 Table1 的 Active 属性值, 该属性值默认为 false, 当设置为 true 时则激活数据库连接, 使得相应的数据表在设计阶段就可以显示数据, 如图 7.4 所示。

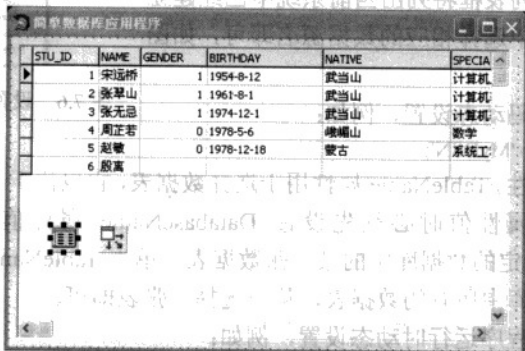


图 7.4 在设计阶段显示数据

设计完毕后运行该程序, 结果如图 7.5 所示。

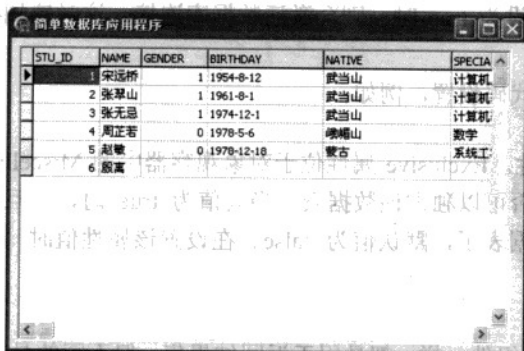


图 7.5 数据库程序运行界面

可以看到, 在工程 project1 中不要任何的 SQL 语句, 而只用三个数据库组件就可以完成非常漂亮的数据显示。足由此可见数据库组件在数据库应用程序开发中的作用和魅力。为此, 将

在下一节中较为系统地介绍一些常用数据库组件,然后进一步说明数据库应用程序开发的相关技术。

7.2.2 常用的 BDE 组件及相关的数据库组件

BDE 组件位于工具面板中的 BDE 标签页上,这类组件通常又称为数据集(类型)组件。其中,常用的组件包括 Table、Query、DataSource 及 DBGrid 等,以下分别介绍。

1. Table 组件

Table 组件是用于表达数据表中的数据和结构的组件,它有四个重要的属性,即 DatabaseName、TableName、Active 和 Exclusive 属性。

(1) DatabaseName 属性。该属性用于选择数据源名,即指定 Table 组件链接的数据源,从而确定了它要与哪个数据库连接。它位于对象观察器的 Databases 标签页中,其设置方法是:找到 DatabaseName 属性项并单击其右边的下拉列表框,这时该列表框将列出当前系统中已经建立的数据源名,只要从中选择相应的数据源名即可,如图 7.6 所示。

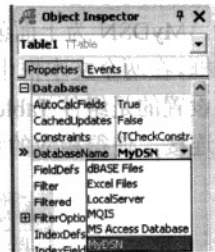


图 7.6 设置 DatabaseName 属性值

该属性值可以用代码动态设置,例如:

```
table1.DatabaseName := 'MyDSN';
```

(2) TableName 属性。TableName 属性用于选择数据表,它也位于对象观察器的 Databases 标签页中。在设置该属性值时必须先设置 DatabaseName 属性值,因为它是用于选择 DatabaseName 属性值指定的数据库中的某一张数据表。单击 TableName 属性项右边的下拉列表框,将列出相应数据库中的所有数据表,从中选择一张表即可。

该属性值也可以在程序运行时动态设置,例如:

```
table1.TableName := 'dbo.stu';
```

(3) Active 属性。Active 属性用于激活数据库连接,一般是在所有数据库组件设置完毕后才设置该属性值。它位于对象观察器中的 Miscellaneous 标签页上,是一个布尔类型,默认值为 false。当把它的值设为 true 时,则会激活数据库连接,这时感知类型的数据库组件将显示数据库中的数据。

该属性值也可以用代码设置,例如:


```
table1.Active := true;
```

(4) Exclusive 属性。Exclusive 属性位于对象观察器中的 Miscellaneous 标签页上,为布尔类型。它用于确定是否可以独占该数据表。当其值为 true 时,一旦程序打开了数据表,其他程序就不能打开该数据表了,默认值为 false。在设置该属性值时,必须关闭数据库连接,否则会出现异常。

2. Query 组件

Query 组件与 Table 组件一样,都是用于返回结果集。但不同的是,Table 组件操作的数据表是相对固定的,而 Query 组件则使用 SQL 语句来对数据表进行操作,不但返回的结果集比较灵活,而且对操作的数据表变动也较大。这从它们的属性对比上看得出,Query 组件也有 DatabaseName 属性,用于指定数据库;但 Query 组件没有 TableName 属性,即它不用静态指

定数据表名,而是在SQL语句中动态指定,这使得Query组件不限于某个数据表上,而是可以灵活地对数据库中所有的表进行操作。

Query组件另一个重要的属性是SQL属性,也位于对象观察器的Databases标签页中。SQL属性是TString类型,其设置方法是:单击SQL属性项右边的省略号按钮,则弹出SQL语句列表编辑器,在此可以输入任意合法的SQL语句(详见第6章的介绍),如图7.7所示。

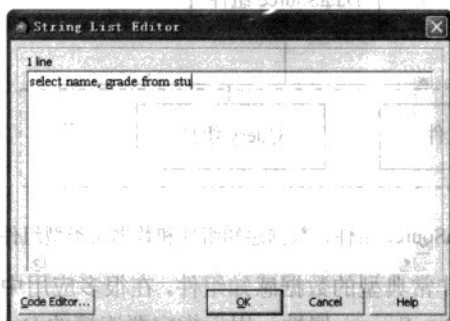


图 7.7 SQL 语句列表编辑器

也可以在程序中动态设置,使得应用起来更为灵活。例如:

```
query1.SQL.Add('select name, grade from stu');  
query1.Active := true;
```

在程序运行时,有时需要动态查询,要传递参数,这时可以使用Params属性作为参数赋值。实际上,在程序运行时Delphi会自动建立名为Params的数组,该数组以0下标开始,依次与动态SQL语句中的参数相对应。例如,观察下面的SQL语句:

```
str:='张翠山';  
query1.Close;  
query1.SQL.Clear;  
query1.sql.Add('select * from student where name = :Param0');  
query1.Params[0].AsString := str;  
query1.open;
```

在上面的代码段中,有一个参数Param0(注意:参数的引用方式为:冒号+参数名,即Param0),同时给数组Params中的第一个元素Params[0]赋字符串变量str的值,该值就传给了变量var1。实际上相当于执行下列查询语句:

```
'select * from student where name = '张翠山';
```

3. DataSource 组件

DataSource组件主要是在数据集组件和感知组件之间起中介作用,通过它可以使多种数据集类型组件(如Query、Table等)和数据感知组件(如DBGrid等)有机而灵活地结合起来,构造丰富多彩的数据显示方式。这几类组件之间的关系可以用图7.8来表示。

DataSource组件的属性不多,主要用到的是DataSet属性,它用来指定DataSource组件连接的数据集类型组件,如Table组件和Query组件等。

4. DBGrid 组件

DBGrid组件位于工具面板中的Data Controls标签页上。该标签页上的组件一般是用于显示数据的,所以通常称为感知组件。

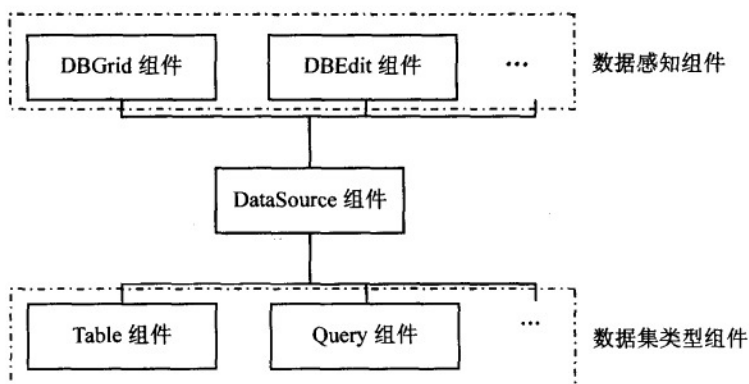


图 7.8 DataSource 组件、数据感知组件和数据集合类型组件之间的关系

其中, DBGrid 组件是非常典型的数据感知组件, 在很多应用中都用这个组件来显示数据。它的一个重要的属性就是 DataSource 属性, 用于指定要连接的 DataSource 组件。

除 DBGrid 组件以外, DBEdit 等组件也是这种类型, 它们的设置基本相同 (有的可能要设置到字段级), 在此不再赘述。

7.2.3 数据库连接的打开和关闭

从图 7.8 中可以看出, 要连接数据库必须把数据集合类型组件、DataSource 组件和数据感知组件之间连接好。具体操作可分为以下几步来进行:

(1) 对于数据集合类型组件, 主要设置它的 DatabaseName 属性和 TableName 属性, 以指定要打开的数据库及库中的数据表。

(2) 对于 DataSource 组件, 要设置它的 DataSet 属性, 以指定 DataSource 组件要连接的数据集合类型组件。

(3) 对于数据感知组件, 要设置它的 DataSource 属性, 有的还需设置 DataField 组件, 它们分别用于指定数据感知组件要连接的 DataSource 组件和相关的字段。

(4) 在完成上述组件属性的设置以后, 还要激活刚设置的数据库连接, 方法是把数据集合类型组件的 Active 属性的值设为 true。

经过上面几个步骤以后, 在感知组件中将显示相应数据库中数据表的内容。

实际上, 在程序运行时也可以动态设置以上相关的属性值, 例如:

```

table1.DatabaseName := 'MyDSN';
table1.TableName := 'dbo.stu';      } 步骤 (1)

DataSource1.DataSet := table1;      } 步骤 (2), 注意, table1 不能写成字符串的形式 'table1'

DBGrid1.DataSource := DataSource1;  } 步骤 (3), 注意, DataSource1 也不能写成字符串的形式 'DataSource1'

table1.Active := true;              } 步骤 (4), 激活连接
  
```

其中, 步骤 (4) 也是通常所称的打开数据库连接, 下面是其等价的语句:

```
table1.open;
```

对数据库访问完毕后还要关闭它，可以采用下列语句：

```
table1.Active := false;
```

或者

```
table1.close;
```

7.3 熟悉数据库的浏览和查询设计

浏览和查询数据几乎都可以利用 Table 组件提供的方法来完成，所以本节将主要介绍 Table 组件常用于数据浏览和查询的一些方法。由于 Query 组件的方法与 Table 组件的方法基本相同，所以就不单独介绍 Query 组件的方法了。

7.3.1 Query 组件提供的方法

数据浏览和查询的过程实际上就是利用 Table 组件的方法来移动记录指针的过程。这些方法的作用和使用方式如表 7.2 所示。

表 7.2 Table 组件的常用方法

方法	作用	例子
First()	使记录指针移到第一条记录	table1.First();
Next()	使记录指针移到当前记录的下一条记录	table1.Next();
Prior()	使记录指针移到当前记录的前一条记录	table1.Prior();
Last()	使记录指针移到最后一条记录	table1.Last();
MoveBy(n:integer)	使记录指针移到当前记录后的第 n 条记录，如果 n=1，它相当于 Next()方法；如果 n=-1，它相当于 Prior()方法	table1.MoveBy(2)

需要注意的是，对于上述方法，每执行一次都会检测和设置 Table 组件的 Bof 和 Eof 属性。当指针指向第一条记录时，如果还要执行一次 First()方法，那么 Bof 将被置为 true，其他情况下 Bof 被置为 false；或者执行 First()方法时，Bof 也被置为 true。

当指针指向最后一条记录时，如果还要执行一次 Last()方法，那么 Eof 将被置为 true，其他情况下 Eof 被置为 false；或者执行 Last()方法时，Eof 也被置为 true。

对于 MoveBy(n:integer)方法，如果移到了第一条记录就设置 Bof 为 true（超过部分不再移到），如果移到了最后一条记录，则置 Eof 为 true（超过部分不再移到）。

如果要访问当前行某一系列的数据时，可用 Table 组件的 table1.FieldValues['列名']方法，该方法是一种返回类型为可变结构的函数，可以赋给 String 变量用于显示。例如下面语句则把当前行的 birthday 列处的数据显示在 Edit1 对象中：

```
Edit1.Text := table1.FieldValues['birthday'];
```

另外，Table 组件的 RecordCount 属性和 FieldCount 属性分别返回数据表记录的行数和列数，它们在编程时也经常用到。

7.3.2 一个数据浏览实例

下面的例子是一个浏览数据库中数据的程序，其中用到 DBGrid、DBEdit 等感知组件，具体步骤如下：

(1) 创建一个新的 VCL 工程，命名为 ViewData.dpr。然后在窗体上放入 Table、DataSource、DBGrid、GroupBox、Edit 组件各一个，放入 Label 组件七个，DBEdit 组件六个，SpeedButton 组件五个。

(2) 设置各组件属性，主要是对组件进行绑定。设置结果如表 7.3 所示。

表 7.3 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Table	Table1	DatabaseName	MyDSN
		TableName	dbo.STU
		Active	true
DataSource	DataSource1	DataSet	Table1
DBGrid	DBGrid1	DataSource	DataSource1
GroupBox	GroupBox1	Caption	(空值)
Edit	Edit1	Caption	(空值)
Label	Label1	Caption	姓名
		Font.Size	10
	Label2	Caption	性别
		Font.Size	10
	Label3	Caption	出生日期
		Font.Size	10
	Label4	Caption	出生地址
		Font.Size	10
	Label5	Caption	专业
		Font.Size	10
	Label6	Caption	成绩
		Font.Size	10
	Label7	Caption	条记录
		Font.Size	8
DBEdit	DBEdit1	DataSource	DataSource1
		DataField	NAME
	DBEdit2	DataSource	DataSource1
		DataField	GENDER
	DBEdit3	DataSource	DataSource1
		DataField	BIRTHDAY

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DBEdit	DBEdit4	DataSource	DataSource1
		DataField	NATIVE
	DBEdit5	DataSource	DataSource1
		DataField	SPECIALITY
	DBEdit6	DataSource	DataSource1
		DataField	GRADE
SpeedButton	SpeedButton1	Caption	第一条记录
	SpeedButton2	Caption	下一条记录
	SpeedButton3	Caption	上一条记录
	SpeedButton4	Caption	最后一条记录
	SpeedButton5	Caption	移 动
Form	Form1	Caption	数据浏览、查询程序

(3) 属性设置完毕后, 对各组件的大小和位置作适当的调整, 可得到窗体设计界面, 如图 7.9 所示。

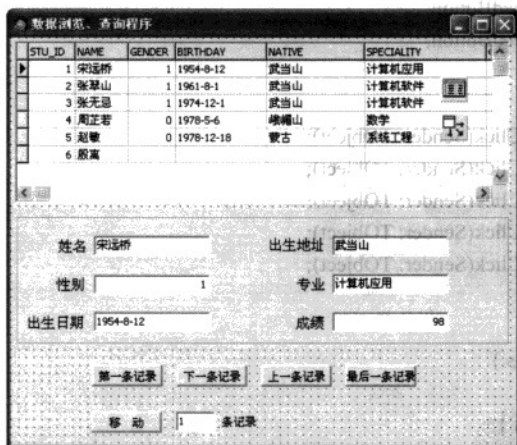


图 7.9 窗体设计界面

(4) 窗体设计完毕以后, 利用 Table 组件的属性和方法为五个按钮添加代码, 工程 ViewData.dpr 的单元文件的代码如下:

```
unit Unit1;  
interface  
uses
```

```
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
Dialogs, Grids, DBGrids, DB, DBTables, StdCtrls, Mask, DBCtrls, ExtCtrls,  
Buttons;
```

```

type
    TForm1 = class(TForm)
        DataSource1: TDataSource;
        DBGrid1: TDBGrid;
        Table1: TTable;
        GroupBox1: TGroupBox;
        Label1: TLabel;
        Label2: TLabel;
        Label3: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        DBEdit1: TDBEdit;
        DBEdit2: TDBEdit;
        DBEdit3: TDBEdit;
        DBEdit4: TDBEdit;
        DBEdit5: TDBEdit;
        DBEdit6: TDBEdit;
        SpeedButton1: TSpeedButton;
        SpeedButton2: TSpeedButton;
        SpeedButton3: TSpeedButton;
        SpeedButton4: TSpeedButton;
        SpeedButton5: TSpeedButton;
        Edit1: TEdit;
        Label7: TLabel;
    procedure SpeedButton5Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);

    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin

```

```
table1.First();
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
    table1.Next();
    if table1.Eof then ShowMessage('已经是最后一条记录，不能再后移!');
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
    table1.Prior();
    if table1.Bof then ShowMessage('已经是第一条记录，不能再前移!');
end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
    table1.Last();
end;

procedure TForm1.SpeedButton5Click(Sender: TObject);
begin
    table1.MoveBy(StrToInt(Edit1.Text))
end;
end.
```

(5) 在窗体设计和代码编写完成以后，就可以执行该程序了。执行结果如图 7.10 所示。

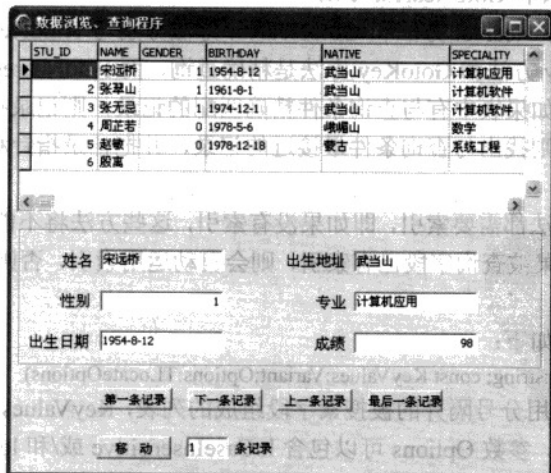


图 7.10 程序 ViewData.dpr 运行界面

在执行的程序中，通过单击“第一条记录”、“下一条记录”、“上一条记录”、“最后一条记录”、“移动”按钮的方法可以有效地实现数据的浏览功能。

7.3.3 数据查询设计

数据查询是数据库应用程序的主要功能之一，几乎在每一个数据库应用程序中都有查询功能。下面介绍几种常用的查询方法。

1. KindKey 方法

在调用 KindKey 方法之前，首先要确定是对哪一列值进行查找。这要通过设置 Table 组件的 IndexFieldName 属性值来完成，然后才能调用 KindKey 方法。如果查询成功，则记录指针移到关键词所在的记录（如果有多个记录含有该关键词，则移到第一个记录）。

例如，如果对 stu 表按姓名查询，可用下列语句来完成：

```
str := Edit1.Text; //str 为 string 类型
table1.IndexFieldName := 'name';
if not table1.FindKey([str]) then ShowMessage('没找到');
```

2. GotoKey 方法

在调用该方法前，首先要建立关于待查字段的索引，然后调用 SetKey 方法，把数据集切换到查询模式，接着要设置被查找的字段值，最后才调用 GotoKey 方法进行查找。

例如，如果改用 GotoKey 方法来完成前面的例子，则可以用下列代码实现：

```
with table1 do
begin
    SetKey; //设置为查询模式
    FieldByName('name').AsString := Edit1.Text; //设置 Edit1 对象的 Text 属性值为被查找的字段值
    if not GotoKey then ShowMessage('没找到'); //进行查找
end;
```

在执行上面的语句之前，必须先建立关于 name 字段的索引，否则会发生错误。

3. FindNearest 方法和 GotoNearest 方法

在调用方法上，FindNearest 方法和 GotoNearest 方法分别与 KindKey 方法和 GotoKey 方法一样。但是，KindKey 方法和 GotoKey 方法是精确查询，而 FindNearest 方法和 GotoNearest 方法是最临近查询，即如果表中有与查询条件精确匹配的记录，则记录指针将移到该记录上；如果没有精确匹配时，则找出与查询条件最接近的记录，并把记录指针移动到该记录上。

4. Locate 方法

上面介绍的查询方法都需要索引，即如果没有索引，这些方法将不能使用。Locate 方法却没有这方面的限制。如果被查询字段已有索引，则会自动运用索引，否则将进行普通查询，但这时效率比较低。

Locate 方法的原型如下：

```
Locate(const KeyFields:string; const KeyValues:Variant;Options:TLocateOptions)
```

其中，KeyFields 是用分号隔开的被搜索字段组成的列表，KeyValues 是用逗号隔开的待查询的字段值组成的列表，参数 Options 可以包含 loCaseInsensitive 或/和 loPartialKey。如果包含 loCaseInsensitive，则表示在查询时不区分大小写；如果包含 loPartialKey，则表示用最临近查询，否则用精确查询。例如：

```
str := edit1.Text; //str 为 string 类型
```

```
if not table1.Locate('name',str,[]) then ShowMessage('没找到'); // 精确查询，对大小写敏感
```

```

if not table1.Locate('stu_id;name',VarArrayOf([2,'张翠山']),[loCaseInsensitive])
then ShowMessage('没找到'); // 精确查询,对大小写不敏感
if not table1.Locate('stu_id;name',VarArrayOf([2,'aa']),[loPartialKey,loCaseInsensitive])
then ShowMessage('没找到');
// 最临近查询,对大小写不敏感

```

7.4 熟悉数据库的更新设计

数据库的更新主要包括数据插入、删除和修改等三种操作,以下分别讲述。

7.4.1 数据的插入

在 Delphi 中,数据的插入有几种方法,它们都是 Table 组件提供的方法。

1. InsertRecord 方法

InsertRecord 方法的语法格式如下:

```
table1.InsertRecord([字段值列表]);
```

例如,利用下列语句可以把“小昭”的有关信息插入数据表 stu 当中:

```
table1.InsertRecord([8, '小昭', 0, StrToDateTime('1980-9-24'),
'波斯', '计算机软件', 85, '痴情, 聪明, 城府深']);
```

需要注意的是,字段值列表中的字段值类型要与数据表中相应的字段类型一致。字段值列表中的字段个数可以少于数据表中的字段数,但出现的部分要依次对应。例如,下列语句是合法的:

```
table1.InsertRecord([8, '小昭']);
```

但是欲用下列语句输入有关“小昭”的学号、姓名和成绩,则是错误的:

```
table1.InsertRecord([8, '小昭', 85]);
```

改进的方法是用 NULL 来“填补”,例如:

```
table1.InsertRecord([8, '小昭', null, null, null, null, 85]);
```

2. Insert...post 方法

Insert...post 方法较 InsertRecord 方法灵活,不必一次性地把所有的插入值作为插入函数参数,从而提高程序的可读性。

该方法的语法格式为:

```
Table1.Insert;
```

```
Table1.FieldName('字段名 1').xxx := 字段值 1;
```

```
...
```

```
Table1.FieldName('字段名 n').xxx := 字段值 1;
```

```
Table1.Post;
```

其中,“xxx”由字段的数据类型来决定;“Table1.Insert”使数据集进入插入状态,它首先在记录指针所指的位置创建一个空记录(其他记录后移),并把该空记录设置为当前记录,然后由其下面的语句输入相应的字段值;“Table1.Post”则用于把数据从内存区写到数据库文件中去,实现真正的数据保存。

例如,如果希望把“小昭”的学号、姓名和成绩输入数据库,则可运用下列几个语句来完成:


```
Table1.Insert;  
Table1.FieldName('stu_id').AsInteger :=8;  
Table1.FieldName('name').AsString :='小昭';  
Table1.FieldName('grade').AsFloat := 85;  
Table1.Post;
```

如果使用 with...do 语句, 则会使代码变得更为简单和易懂。例如, 上面语句段等价于下面的一条语句:

```
with Table1 do  
begin  
    Insert;  
    FieldByName('stu_id').AsInteger :=8;  
    FieldByName('name').AsString :='小昭';  
    FieldByName('grade').AsFloat := 85;  
    Post;  
end;
```

3. AppendRecord 方法

AppendRecord 方法与 InsertRecord 方法类似, 它也是把所有待插入的字段值作为方法的参数。但不同的是, AppendRecord 方法首先在数据集的末尾插入记录并调用 post 方法提交数据。例如:

```
table1.AppendRecord([8,  
    '小昭',  
    0,  
    StrToDateTime('1980-9-24'),  
    '波斯',  
    '计算机软件',  
    85,  
    '痴情, 聪明, 城府深']);
```

4. Append...post 方法

该方法与 Insert...post 方法类似, 但不同的是它是在数据集的末尾建立一个新的空记录并把该记录设置为当前记录, 然后用相关语句插入数据, 之后用 post 方法提交数据。例如:

```
with Table1 do  
begin  
    Append;  
    FieldByName('stu_id').AsInteger :=8;  
    FieldByName('name').AsString :='小昭';  
    FieldByName('grade').AsFloat := 85;  
    Post;  
end;
```

7.4.2 数据的删除

Table 组件提供 delete 方法来删除当前记录。如果数据集为空, 则调用该方法时将产生一个异常。其调用格式为:

```
table1.delete;
```

如果要清除表中的所有数据, 则可用下列语句:

```
for i := 0 to table1.RecordCount-1 do table1.delete;
```

7.4.3 数据的修改

一般来说,一些感知类型组件在默认情况下都是处于可编辑状态,当编辑框失去焦点时会自动调用 post 方法提交数据,从而完成数据的修改。也就是说,感知类型组件在默认情况下一般都可进行数据修改。

如果让这些感知类型组件不具有修改功能,只要把它的 ReadOnly 属性值设置为 true 即可。另外,还可以用 Edit 方法实现对当前记录的修改功能。例如:

```
with table1 do
begin
    Edit;
    FieldByName('stu_id').AsInteger :=8;
    FieldByName('name').AsString :='小昭';
    FieldByName('grade').AsFloat := 85;
    Post;
end;
```

7.5 Delphi BDE 数据库的应用实例

下面通过一个具体应用的例子来加深对 BDE 数据库开发的基本知识的理解。这个程序的功能虽然非常简单,但它包含了 BDE 数据库开发的基本步骤。

7.5.1 程序功能设计

在高校中,任课教师在期末考试后需要计算各班(有多个班级)学生的期评成绩,期评成绩是由平时成绩的 30%和期考成绩的 70%相加而得到。此外,还要统计出各个分数段(成绩等级)中的学生数量,即对五个分数段:90~100、80~89、70~79、60~69 及 60 分以下,计算出每个分数段中的学生数量及其占全班总学生数的比例等。

这个程序实现的管理系统就称为学生成绩管理系统。

这里,用 Delphi 来实现这个程序,这个程序可以从中国水利水电出版社网站上下载(含 100%源代码)。另外,为了方便,程序使用了 Microsoft Access 数据库(不用安装数据库),同时还提供已经编译而生成的 exe 文件,以方便教师或者感兴趣的读者直接使用。

该系统可以划分为以下几个子系统:

- 成绩输入子系统。用于输入和修改相关数据,包括学生学号、姓名、平时成绩、期考成绩以及有关学生的班级信息等。
- 生成期评成绩子系统。期评成绩作为学生一个学期的最终成绩,它是由平时成绩和期考成绩按照一定比例来构成的,该子系统正是用于自动生成期评成绩的一个系统模块。
- 成绩删除子系统。用于删除学生记录信息。
- 成绩查询与统计子系统。实现按名字或出生地址等字段查询,统计各分数段的学生数等。

7.5.2 数据库设计

从 Windows 系统中打开 Microsoft Access，创建名为 ResultDB.mdb 的 Access 数据，在其中创建如表 7.4 所示的一张表，取名为 student，结果如图 7.11 所示。

表 7.4 学生成绩信息表

字段名	数据类型	字段大小	说明
stu_id	文本	10	学号
name	文本	8	姓名
pea_resu	数字	1 位小数位	平时成绩
fin_resu	数字	1 位小数位	期考试成绩
ave_resu	数字	1 位小数位	期评成绩
class	文本	10	所在班级

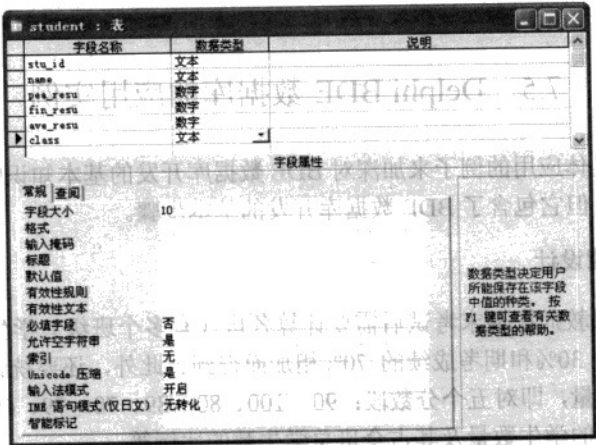


图 7.11 表 student 的设计结构

7.5.3 创建 ODBC 数据源

关于 SQL Server 数据库的 ODBC 数据源的创建方法已在前文进行了介绍。本节使用的是 Access 数据库，然而不管是什么数据库，其创建 ODBC 数据源的方法基本相同。为了完整起见，在此对于 Access 数据库的数据源创建方法作一个简单的介绍。

创建名为 AccessDSN 的数据源可按照下列步骤完成：

- (1) 打开 ODBC 数据源管理器，并选择 System DSN 标签页。
- (2) 在 ODBC 数据源管理器中，单击 Add 按钮，打开选择数据源驱动程序对话框。在此对话框中选择 Access 数据库驱动程序，然后单击“确定”按钮，将弹出选择数据库对话框。
- (3) 在选择数据库对话框中单击“选择”按钮，在弹出的对话框中找到并选择上面已创建的数据库名——ResultDB.mdb，然后单击“确定”按钮，将在 ODBC 数据源管理器中看到已经创建完毕的数据源 AccessDSN，如图 7.12 所示。

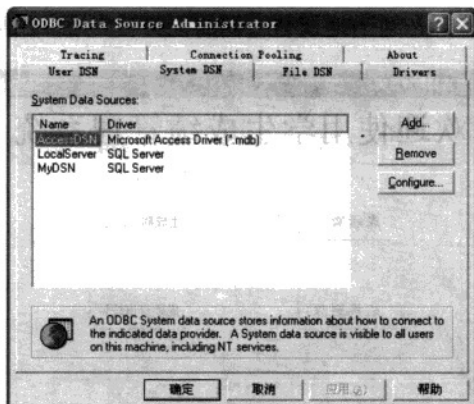


图 7.12 ODBC 数据源管理器 (已创建 AccessDSN 数据源)

(4) 单击 ODBC 数据源管理器中的“确定”按钮，则 Access 数据库的数据源创建完毕。

7.5.4 创建 Delphi 应用程序

在数据库和数据源创建完毕以后，接下来就要创建 Delphi 应用程序，然后要设计各子系统主界面对应的窗体，最后设计各模块的实现代码。

首先在 Delphi 2005 中创建一个名为 ResultManage.dpr 的工程，其对应的单元文件为 Unit1.pas，并把单元文件对应的窗体作为程序的主窗体。在主窗体上分别添加 Label 组件、GroupBox 组件和 Button 组件各一个，SpeedButton 组件四个。对窗体及各组件属性值的设置情况如表 7.5 所示。

表 7.5 窗体及各属性值的设置情况

组件类别	组件名称	属性设置项目	设置结果
Label	Label1	Caption	欢迎使用学生成绩管理系统
		Font.Size	30
		Font.Style	[fsBold]
GroupBox	GroupBox1	Caption	(空值)
SpeedButton	SpeedButton1	Caption	生成期评成绩
		Font.Size	10
	SpeedButton2	Caption	成绩删除
		Font.Size	10
	SpeedButton3	Caption	成绩查询统计
		Font.Size	10
	SpeedButton4	Caption	成绩输入
		Font.Size	10
Button	Button1	Caption	退出
Form	Form1	Caption	学生成绩管理系统

各组件属性值设置完毕后，适当地调整它们的位置和大小，结果如图 7.13 所示。

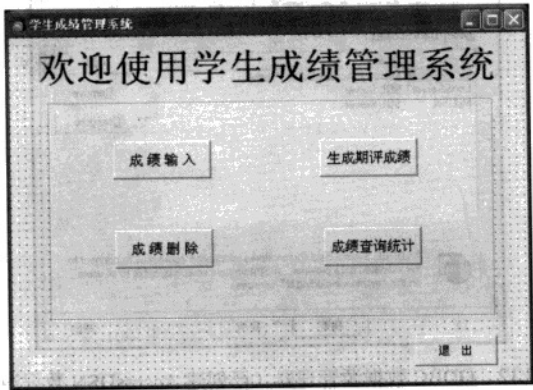


图 7.13 程序 ResultManage.dpr 主窗体的设计效果

经过上述窗体设计后，应用程序的主界面已经“初具规模”，但它还没有任何功能，为此还必须实现其包含的子功能模块。

设计的基本思想是：对每一个子系统（子功能模块），用一个窗体来实现。因此，需要再创建四个窗体。创建的方法是：在 Delphi 2005 IDE 中，选择 File→New→Form 菜单命令，则生成名为 Form2 的窗体（程序主窗体名为 Form1），在此选择使用默认的名称；然后用同样的方法，继续创建余下的三个窗体，分别采用默认的名称 Form3、Form4、Form5，得到的工程文件结构如图 7.14 所示。

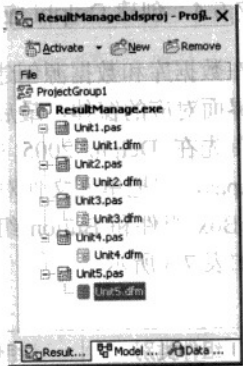
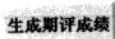

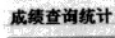


图 7.14 工程 ResultManage.dpr 的管理器

图 7.13 所示的程序主窗体上的按钮用于打开相应的子系统界面，这些按钮与子系统界面（Form）的对应关系如表 7.6 所示。

表 7.6 按钮、窗体和子系统的对应关系

按 钮		窗体（名称）	子系统
按钮图形	按钮名称		
	SpeedButton4	Form2	生成期评成绩
	SpeedButton2	Form3	成绩删除
	SpeedButton3	Form4	成绩查询统计
	SpeedButton1	Form5	成绩输入

在程序运行时，默认所有的窗体（Form1~Form5）都被创建了，但是看到的只有 Form1。可以通过编写按钮事件处理过程代码来显示相应的窗体（子系统界面），方法是调用 Form 的 Show 方法来显示。窗体 Form1 对应的单元文件 Unit1.pas 的代码如下：

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Buttons, StdCtrls, Unit2, Unit3, Unit4, Unit5, DB, DBTables;

type
  TForm1 = class(TForm)
    GroupBox1: TGroupBox;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    SpeedButton3: TSpeedButton;
    SpeedButton4: TSpeedButton;
    Button1: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure SpeedButton4Click(Sender: TObject);
    procedure SpeedButton3Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.SpeedButton1Click(Sender: TObject);
begin
  Form2.Show;
end;

procedure TForm1.SpeedButton2Click(Sender: TObject);
begin
  Form3.Show;
end;

procedure TForm1.SpeedButton3Click(Sender: TObject);
begin
```

```

Form4.Show;
end;

procedure TForm1.SpeedButton4Click(Sender: TObject);
begin
    Form5.Show;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    self.Close;
end;
end.

```

需要在 Unit1.pas 中的代码时要在 uses 部分加上 Unit2、Unit3、Unit4 及 Unit5，否则会产生编译错误。

7.5.5 成绩输入子系统的设计与实现

该子系统用于输入学生的有关信息，包括学号、姓名、平时成绩、期考成绩、期评成绩及学生所在的班级名称等。如果利用 DBEdit 组件，则很容易实现这一输入功能。但为更全面地学习，在这里介绍使用一般的编辑框组件——Edit 组件，来实现这一输入功能。同时为了看到插入数据的实时结果，还使用 DBGrid 组件来显示数据，但在这里 DBGrid 组件仅用于显示数据，而不具有编辑功能（其 ReadOnly 属性值设置为 true）。

基于以上思想，需要在窗体 Form5 上放入以下组件：DBGrid、DataSource、Table、GroupBox 组件各一个，Edit 和 Label 组件各六个，Button 组件四个。各组件的属性设置情况如表 7.7 所示。

表 7.7 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DBGrid	DBGrid1	DataSource	DataSource1
DataSource	DataSource1	DataSet	Table1
Table	Table1	DatabaseName	AccessDSN
		TableName	student
		Active	true
GroupBox	GroupBox1	Caption	(空值)
Edit	对于组件 Edit1、Edit2、Edit3、Edit4 和 Edit6，它们均采用默认的属性值，而 Edit5 的 Enabled 属性值则设为 false，只用于显示期评成绩，不能输入		
Label	Label1	Caption	学号
	Label2	Caption	姓名
	Label3	Caption	平时成绩
	Label4	Caption	期考成绩
	Label5	Caption	期评成绩
	Label6	Caption	班级

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Button	Button1	Caption	保存数据
	Button2	Caption	在当前位置插入
	Button3	Caption	在末尾插入
	Button4	Caption	退出
Form	Form5	Caption	成绩输入

对于 Table 组件, 要依次设置它的 DatabaseName、TableName 和 Active 属性, 因为只有选择了数据库 (通过数据源) 才能找到相应的数据表并作出选择, 才能进一步激活数据连接。属性设置完成后, 调整各组件的大小和位置, 成绩输入子系统的界面如图 7.15 所示。

stu_id	name	pea_resu	fin_resu	ave_resu	class
0538010101	宋远桥	98	98	0	计科043
0538010102	张翠山	97	89	0	计科044
0538010103	张无忌	92	97	0	计科044
0538010104	周芷若	60	69	0	计科044
0538010105	赵敏	78	88	0	计科044
0538010106	俞莲舟	85	90	0	计科044

学号	Edt1	期考成绩	Edt4
姓名	Edt2	期评成绩	Edt5
平时成绩	Edt3	班级	Edt6

在当前位置插入 在末尾插入 保存数据 退出

图 7.15 成绩输入子系统界面 (Form5 窗体)

编码时, 由于采用了 Edit 组件作为输入框, 所以很多组件的可用性要用代码来控制。这样, 虽然讲述过程比较长, 但是从中可以学到很多有用的知识。下列代码是窗体 Form5 对应的单元文件代码 (在单元文件 Unit5.pas 中)。

```
unit Unit5;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, DB, DBTables, StdCtrls, Mask, DBCtrls, ExtCtrls;

type
  TForm5 = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
```



```
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Button1: TButton;
Button2: TButton;
Button3: TButton;
Button4: TButton;
DBGrid1: TDBGrid;
DataSource1: TDataSource;
Table1: TTable;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Edit4: TEdit;
Edit5: TEdit;
Edit6: TEdit;
procedure Button3Click(Sender: TObject);
procedure Edit6MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure Edit4MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure Edit3MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure Edit2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure Edit1MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
procedure DBGrid1MouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
procedure FormCreate(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button1Click(Sender: TObject);

private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form5: TForm5;

implementation
```

```
{SR *.dfm}
```

```
procedure TForm5.Button1Click(Sender: TObject); //保存数据
```

```
begin
```

```
    table1.Edit;
```

```
    Table1.FieldName('stu_id').AsString := Edit1.Text;
```

```
    Table1.FieldName('name').AsString := Edit2.Text;
```

```
    Table1.FieldName('pea_resu').AsFloat := StrToFloat(Edit3.Text);
```

```
    Table1.FieldName('fin_resu').AsFloat := StrToFloat(Edit4.Text);
```

```
    Table1.FieldName('ave_resu').AsString := Edit5.Text;
```

```
    Table1.FieldName('class').AsString := Edit6.Text;
```

```
    table1.Post;
```

```
    button1.Enabled := false;
```

```
    button2.Enabled := true;
```

```
    button3.Enabled := true;
```

```
    DBGrid1.Enabled := true;
```

```
    Edit1.Enabled := false; Edit1.Text := '';
```

```
    Edit2.Enabled := false; Edit2.Text := '';
```

```
    Edit3.Enabled := false; Edit3.Text := '';
```

```
    Edit4.Enabled := false; Edit4.Text := '';
```

```
    Edit6.Enabled := false; Edit6.Text := '';
```

```
end;
```

```
procedure TForm5.Button2Click(Sender: TObject); //在当前位置插入
```

```
begin
```

```
    table1.Insert;
```

```
    Edit1.Text := '输入学号'; Edit1.Enabled := true;
```

```
    Edit2.Text := '输入姓名'; Edit2.Enabled := true;
```

```
    Edit3.Text := IntToStr(0); Edit3.Enabled := true;
```

```
    Edit4.Text := IntToStr(0); Edit4.Enabled := true;
```

```
    Edit5.Text := '不能输入'; //Edit1.Enabled := true;
```

```
    Edit6.Text := '输入班级'; Edit6.Enabled := true;
```

```
    button2.Enabled := false;
```

```
    button1.Enabled := true;
```

```
    button3.Enabled := false;
```

```
    DBGrid1.Enabled := false;
```

```
end;
```

```
procedure TForm5.Button4Click(Sender: TObject); //关闭子系统
begin
    self.close;
end;

procedure TForm5.FormCreate(Sender: TObject); //初始化各组件
begin
    Edit1.Text := Table1.FieldByName('stu_id').AsString;
    Edit2.Text := Table1.FieldByName('name').AsString;
    Edit3.Text := Table1.FieldByName('pea_resu').AsString;
    Edit4.Text := Table1.FieldByName('fin_resu').AsString;
    Edit5.Text := Table1.FieldByName('ave_resu').AsString;
    Edit6.Text := Table1.FieldByName('class').AsString;

    Edit1.Enabled := false; //禁用各 Edit 组件
    Edit2.Enabled := false;
    Edit3.Enabled := false;
    Edit4.Enabled := false;
    Edit6.Enabled := false;

    button1.Enabled := false;
end;

procedure TForm5.DBGrid1MouseUp(Sender: TObject; Button: TMouseButton;
//使得单击数据记录时, 编辑框自动显示相应的记录
    Shift: TShiftState; X, Y: Integer);
begin
    Edit1.Text := Table1.FieldByName('stu_id').AsString;
    Edit2.Text := Table1.FieldByName('name').AsString;
    Edit3.Text := Table1.FieldByName('pea_resu').AsString;
    Edit4.Text := Table1.FieldByName('fin_resu').AsString;
    Edit5.Text := Table1.FieldByName('ave_resu').AsString;
    Edit6.Text := Table1.FieldByName('class').AsString;
end;

procedure TForm5.Edit1MouseMove(Sender: TObject; Shift: TShiftState; X,
//鼠标移到输入用的编辑框上面时, 框中内容变成被选中状态, 以便于编辑
    Y: Integer);
begin
    Edit1.SelectAll;
end;

procedure TForm5.Edit2MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
```

```
    Edit2.SelectAll;
end;

procedure TForm5.Edit3MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Edit3.SelectAll;
end;

procedure TForm5.Edit4MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Edit4.SelectAll;
end;

procedure TForm5.Edit6MouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
    Edit6.SelectAll;
end;

procedure TForm5.Button3Click(Sender: TObject); //在末尾插入
begin
    table1.Append;
    Edit1.Text := '输入学号'; Edit1.Enabled := true;
    Edit2.Text := '输入姓名'; Edit2.Enabled := true;
    Edit3.Text := IntToStr(0); Edit3.Enabled := true;
    Edit4.Text := IntToStr(0); Edit4.Enabled := true;
    Edit5.Text := '不能输入';
    Edit6.Text := '输入班级'; Edit6.Enabled := true;
    button2.Enabled := false;
    button1.Enabled := true;
    button3.Enabled := false;
    DBGrid1.Enabled := false;
end;
end.
```

7.5.6 成绩删除子系统的设计与实现

该子系统用于删除数据库中已经存在的有关学生成绩的数据。有两种删除方式：删除所有数据和删除当前记录。为此，在窗体 Form3 上放入下列组件：DBGrid、DataSource、Table 组件各一个，Button 组件三个。各组件的属性设置情况如表 7.8 所示。

表 7.8 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DBGrid	DBGrid1	DataSource	DataSource1
DataSource	DataSource1	DataSet	Table1
Table	Table1	DatabaseName	AccessDSN
		TableName	student
		Active	true
Button	Button1	Caption	删除当前记录
	Button2	Caption	删除所有记录
	Button3	Caption	退 出
Form	Form3	Caption	成绩 删 除

属性设置完成后，适当地调整组件的大小和位置，尽量使之简洁而美观。结果如图 7.16 所示。

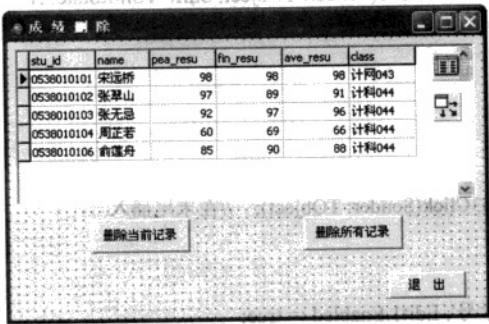


图 7.16 成绩删除子系统界面 (Form3 窗体)

该子系统的实现主要是编写“删除当前记录”按钮和“删除所有记录”按钮的事件处理过程。结果在单元文件 Unit3.pas (对应窗体 Form3) 中生成的代码如下：

```
unit Unit3;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, Grids, DBGrids, StdCtrls;

type
  TForm3 = class(TForm)
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    Table1: TTable;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
  procedure Button3Click(Sender: TObject);
  procedure Button2Click(Sender: TObject);
  end;
```

```
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;

var
Form3: TForm3;

implementation
{$R *.dfm}

procedure TForm3.Button1Click(Sender: TObject); // 删除当前记录
begin
table1.Delete;
end;

procedure TForm3.Button2Click(Sender: TObject); // 删除所有记录
begin
if MessageDlg('确认要删除所有的记录吗?', mtWarning, [mbYes, mbNo], 0) = mrYes then
begin
while not table1.IsEmpty do table1.delete;
end;
end;

procedure TForm3.Button3Click(Sender: TObject);
begin
close;
end;
end.
```

7.5.7 生成期评成绩子系统的设计与实现

学生的期评成绩是由平时成绩和期考成绩按照一定的比例组成的。期评成绩生成子系统就是完成由平时成绩和期考成绩生成期评成绩的一个模块,其中,平时成绩和期考成绩所占的比例(百分数)是可以设定的。其界面如图7.17所示。

stu_id	name	pea_resu	tr_resu	ave_resu	class
0538010101	宋远桥	98		0	计网043
0538010102	张草山	97	89	0	计网044
0538010103	张无息	92	97	0	计网044
0538010104	周正岩	60	69	0	计网044
0538010106	俞耀舟	85	90	0	计网044

平时成绩百分数: 0%

期考成绩百分数: 0%

生成期评成绩

退出

图 7.17 生成期评成绩子系统界面 (Form2 窗体)

该界面是通过在窗体 Form2 上加入以下组件而得到的：DBGrid、DataSource、Table 组件各一个，Label、UpDown、Edit 和 Button 组件各两个。各组件的属性设置情况如表 7.9 所示。

表 7.9 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DBGrid	DBGrid1	DataSource	DataSource1
DataSource	DataSource1	DataSet	Table1
Table	Table1	DatabaseName	AccessDSN
		TableName	student
		Active	true
Edit	对于两个 Edit 组件，它们均采用默认的属性值，名称 (name 属性值) 分别为 Edit1、Edit2		
Label	Label1	Caption	平时成绩百分数：
	Label2	Caption	期考试成绩百分数：
Button	Button1	Caption	生成期评成绩
	Button2	Caption	退 出
UpDown	UpDown1	Max	100
		Min	0
		Position	30
	UpDown2	Max	100
		Min	0
		Position	70
Form	Form2	Caption	生成期评成绩

下列代码是窗体 Form2 对应的单元文件的代码（在单元文件 Unit2.pas 中）：

```

unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Grids, DBGrids, StdCtrls, DB, DBTables, ComCtrls;

type
  TForm2 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    Table1: TTable;
    UpDown1: TUpDown;
    Edit1: TEdit;
    Label1: TLabel;
  end;

```

```

    UpDown2: TUpDown;
    Edit2: TEdit;
    Label2: TLabel;
    procedure Button2Click(Sender: TObject);
    procedure Edit2MouseMove(Sender: TObject; Shift: TShiftState; X,
        Y: Integer);
    procedure Edit1MouseMove(Sender: TObject; Shift: TShiftState; X,
        Y: Integer);
    procedure Button1Click(Sender: TObject);
    procedure Edit2Exit(Sender: TObject);
    procedure Edit1Exit(Sender: TObject);
    procedure UpDown2Click(Sender: TObject; Button: TUDBtnType);
    procedure FormCreate(Sender: TObject);
    procedure UpDown1Click(Sender: TObject; Button: TUDBtnType);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form2: TForm2;

implementation

{$R *.dfm}

procedure TForm2.UpDown1Click(Sender: TObject; Button: TUDBtnType); //设平时成绩的百分数
begin
    Edit1.Text := IntToStr(UpDown1.Position);
    UpDown2.Position := 100-UpDown1.Position;
    Edit2.Text := IntToStr(UpDown2.Position);
end;

procedure TForm2.FormCreate(Sender: TObject); //初始化 Edit1 和 Edit2
begin
    Edit1.Text := IntToStr(UpDown1.Position);
    Edit2.Text := IntToStr(UpDown2.Position);
end;

procedure TForm2.UpDown2Click(Sender: TObject; Button: TUDBtnType); //设期评成绩的百分数
begin
    Edit2.Text := IntToStr(UpDown2.Position);
    UpDown1.Position := 100-UpDown2.Position;
    Edit1.Text := IntToStr(UpDown1.Position);

```



```
end;
```

```
procedure TForm2.Edit1Exit(Sender: TObject);
```

```
//当 Edit1 失去焦点时, 调整组件 UpDown1 和 UpDown2 的 position 属性值,
```

```
//以及 Edit2 的 Text 属性值
```

```
begin
```

```
    UpDown1.Position := StrToInt(Edit1.Text);
```

```
    UpDown2.Position := 100-UpDown1.Position;
```

```
    Edit2.Text := IntToStr(UpDown2.Position);
```

```
end;
```

```
procedure TForm2.Edit2Exit(Sender: TObject);
```

```
//当 Edit2 失去焦点时, 调整组件 UpDown1 和 UpDown2 的 position 属性值,
```

```
//以及 Edit1 的 Text 属性值
```

```
begin
```

```
    UpDown2.Position := StrToInt(Edit2.Text);
```

```
    UpDown1.Position := 100-UpDown2.Position;
```

```
    Edit1.Text := IntToStr(UpDown1.Position);
```

```
end;
```

```
procedure TForm2.Button1Click(Sender: TObject); //生成各学生的期评成绩
```

```
var pea_x, fin_x, ave_x: Real;
```

```
    pea_per, fin_per: integer;
```

```
begin
```

```
    table1.First;
```

```
    while not table1.Eof do
```

```
    begin
```

```
        Table1.Edit;
```

```
        pea_x := table1.FieldValues['pea_resu'];
```

```
        fin_x := table1.FieldValues['fin_resu'];
```

```
        pea_per := StrToInt(Edit1.Text);
```

```
        fin_per := StrToInt(Edit2.Text);
```

```
        ave_x := (pea_per/100.0)*pea_x + (fin_per/100.0)*fin_x;
```

```
        Table1.FieldName('ave_resu').AsFloat := ave_x;
```

```
        Table1.Post;
```

```
        table1.Next;
```

```
    end;
```

```
end;
```

```
procedure TForm2.Edit1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
```

```
//鼠标移到 Edit1 上方时, 其中的数字被选择
```

```
begin
```

```
    Edit1.SelectAll;
```

```
end;
```

```
procedure TForm2.Edit2MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
```

//鼠标移到 Edit2 上方时, 其中的数字被选择

```
begin
    Edit2.SelectAll;
end;

procedure TForm2.Button2Click(Sender: TObject); //关闭窗体 (子系统界面)
begin
    close;
end;
end.
```

7.5.8 成绩查询统计子系统的设计与实现

查询与统计是本程序的主要功能。在此, 要实现按学号、姓名、期评成绩 (当然是生成期评成绩后) 和班级的联合查询。为此, 在成绩查询与统计子系统的主界面窗体 (Form4) 上放入下列组件: DBGrid、DataSource、Query、Memo、ComboBox 组件各一个, Label 组件四个、Edit 和 Button 组件各三个, GroupBox 组件两个。注意, 由于需要动态查询, 所以这里使用了 Query 组件, 而不是 Table 组件。

各组件的属性设置情况如表 7.10 所示。

表 7.10 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DBGrid	DBGrid1	DataSource	DataSource1
DataSource	DataSource1	DataSet	Table1
Query	Query 1	DatabaseName	AccessDSN
		Active	true
		SQL	select * from student
Memo	Memo1	Lines	Memo1
ComboBox	ComboBox1	Text	(空值)
Edit	Edit1	Text	(空值)
	Edit2	Text	(空值)
	Edit3	Text	(空值)
Label	Label1	Caption	学号:
	Label2	Caption	姓名:
	Label3	Caption	成绩等级:
	Label4	Caption	班级:
Button	Button1	Caption	执 行 查 询
	Button2	Caption	显示所有数据
	Button3	Caption	退出

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
GroupBox	GroupBox 1	Caption	设置查询与统计条件:
	GroupBox 2	Caption	查询结果:
Form	Form4	Caption	成绩查询统计

与 Table 组件不同的是, Query 没有 TableName 属性, 它需要在程序运行时进行动态设置, 并且可以设置多种查询条件, 这就增加了数据查询的灵活性。属性设置完后, 调整各组件的大小和位置, 结果成绩查询与统计子系统的界面如图 7.18 所示。

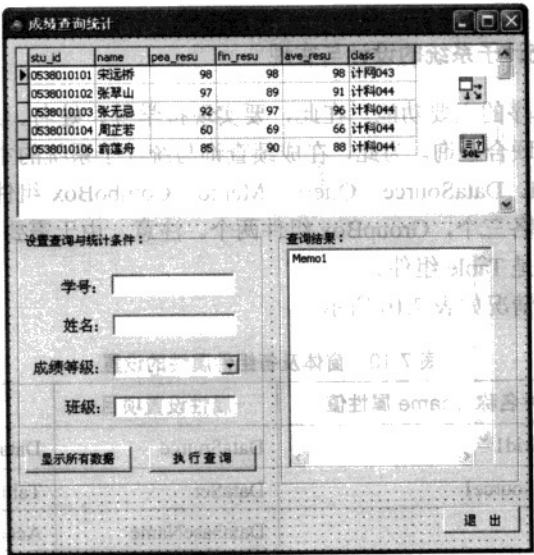


图 7.18 成绩查询与统计子系统的界面 (Form4 窗体)

从图 7.18 所示的界面中可以看出, 该系统可以按四个条件进行联合查询。如果某一条件的查询值为空值 (框中为空值), 则默认为忽略这个查询条件。要完成这些功能, 对 Form4 编写相应的代码, 结果如下 (在单元文件 Unit4.pas 中):

```
unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, DB, DBTables, Grids, DBGrids, StdCtrls;

type
  TForm4 = class(TForm)
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    Query1: TQuery;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
```

```

Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Label4: TLabel;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Button1: TButton;
ComboBox1: TComboBox;
Memo1: TMemo;
Button2: TButton;
Button3: TButton;
procedure Button3Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form4: TForm4;

implementation

{$R *.dfm}

procedure TForm4.Button1Click(Sender: TObject); //实现联合查询
var SqlStr, StrVal, ComStr: string;
    flag: integer;
begin
    Memo1.Clear;
    Memo1.Lines.Add('学号,      姓名      ');
    Memo1.Lines.Add('-----');

    flag:=0;

    StrVal := trim(Edit1.Text);
    if StrVal <> '' then
    begin
        SqlStr := 'select * from student where (stu_id ='+ '''+StrVal+ ''')';
        flag := 1;
    end;
end;

```

```
StrVal := trim(Edit2.Text);
if StrVal <> "" then
begin
    if flag = 0 then
    begin
        SqlStr := 'select * from student where (name ='+ '''+StrVal+ ''')';
        flag := 1;
    end
    else
    begin
        SqlStr := SqlStr + ' and (name = '+ '''+StrVal+ ''')';
    end;
end;

StrVal := trim(Edit3.Text);
if StrVal <> "" then
begin
    if flag = 0 then
    begin
        SqlStr := 'select * from student where (class ='+ '''+StrVal+ ''')';
        flag := 1;
    end
    else
    begin
        SqlStr := SqlStr + ' and (class = '+ '''+StrVal+ ''')';
    end;
end;

StrVal := trim(ComboBox1.Text);

if StrVal = '优秀(90-100)' then ComStr := '(ave_resu >= 90) and (ave_resu <= 100)';
if StrVal = '良好(80-89)' then ComStr := '(ave_resu >= 80) and (ave_resu < 90)';
if StrVal = '中等(70-79)' then ComStr := '(ave_resu >= 70) and (ave_resu < 80)';
if StrVal = '及格(60-69)' then ComStr := '(ave_resu >= 60) and (ave_resu < 70)';
if StrVal = '不及格(60 以下)' then ComStr := '(ave_resu >= 0) and (ave_resu < 60)';

if StrVal <> '(空值)' then
begin
    if flag = 0 then
    begin
        SqlStr := 'select * from student where '+ ComStr;
        flag := 1;
    end
    else
    begin
        SqlStr := SqlStr + ' and ' + ComStr;
    end;
end;
```

```
        end;
    end;

    if flag = 0 then
    begin
        ShowMessage('请选择查询条件');
        exit;
    end;

    query1.Close;
    query1.sql.Text := SqlStr;
    query1.open;

    flag := 0;    //把 flag 用作计数器
    while not query1.Eof do    //显示查询结果（包括学号和姓名）
    begin
        SqlStr := query1.FieldByName('stu_id').AsString;
        SqlStr := SqlStr + ', ' + query1.FieldByName('name').AsString;
        Memo1.Lines.Add(SqlStr);
        query1.Next;
        flag := flag + 1;
    end;
    Memo1.Lines.Add('-----');
    Memo1.Lines.Add('一共有 ' + IntToStr(flag) + ' 项符合查询条件');
end;

procedure TForm4.FormCreate(Sender: TObject);
begin
    ComboBox1.ItemIndex := 0;
end;

procedure TForm4.Button2Click(Sender: TObject);
begin
    query1.Close;
    query1.sql.Text := 'select * from student';
    query1.open;
end;

procedure TForm4.Button3Click(Sender: TObject);
begin
    close;
end;
end.
```

7.5.9 运行程序

代码编写完后，就可以编译、运行这个程序了，其运行界面如图 7.19 所示。

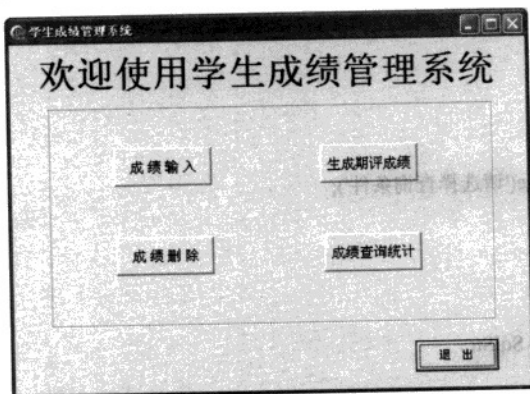


图 7.19 程序 ResultManage 的运行界面

在运行的主界面中单击“成绩输入”按钮，将出现“成绩输入”子系统界面。在此可以输入学生的学号、姓名、班级、期评成绩、平时成绩和期考成绩等有关学生信息，如图 7.20 所示。

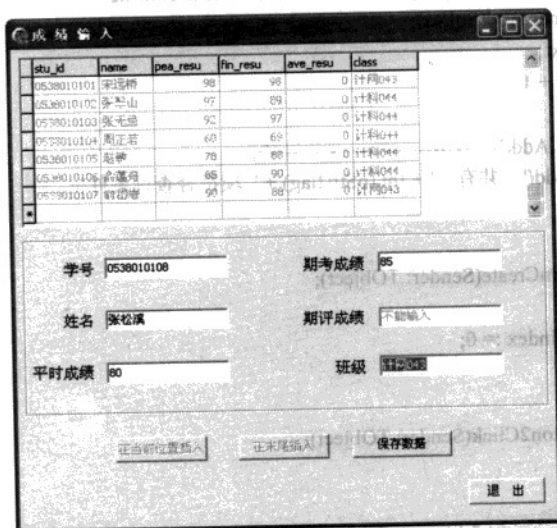


图 7.20 “成绩输入”子系统界面

当在程序主界面中单击“生成期评成绩”按钮时，将出现“生成期评成绩”子系统界面。在此可以设定平时成绩和期考成绩的百分比，然后由此生成期评成绩。图 7.21 表示了如期评成绩中平时成绩占 30%、期考成绩占 70%。

如果在程序主界面中单击“成绩查询统计”按钮时，将出现“成绩查询统计”子系统界面。在此可以按照学号、姓名、成绩等级（优、良、中、及格和不及格）和班级进行查询。如果某一项的关键词为空（框中的内容为空），则默认不把该项作为搜索条件。例如，如果在“成绩等级”对应的下拉列表框中选择“优秀(90-100)”，而把其他的框设置为空，那么程序将默认找出所有期评成绩在 90~100 之间的学生，如图 7.22 所示；如果同时把“班级”对应的文本

框设置为“计科 044”，那么程序将找出所有期评成绩在 90~100 之间的且所在班级为“计科 044”的学生。

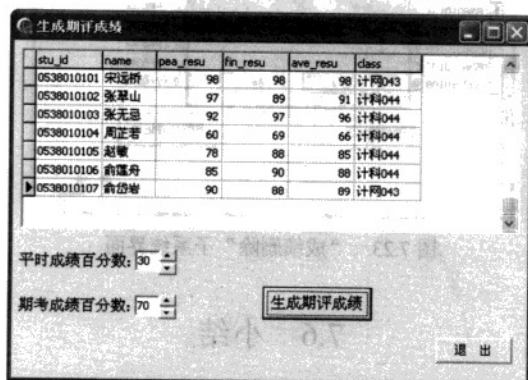


图 7.21 “生成期评成绩”子系统界面

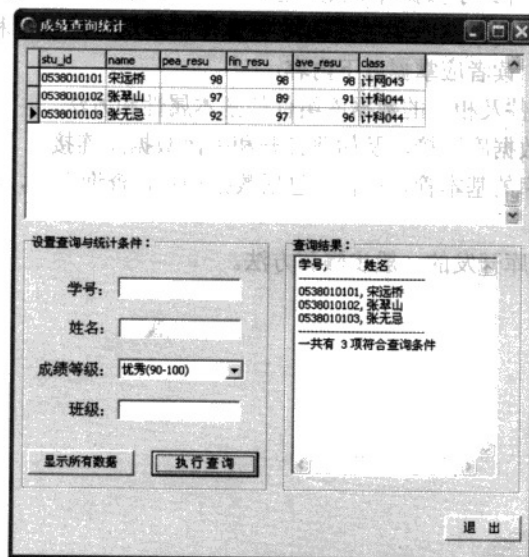


图 7.22 “成绩查询统计”子系统界面

如果在程序主界面中单击“成绩删除”按钮，将出现“成绩删除”子系统界面。该子系统提供了两个删除功能：删除当前记录和删除所有记录。任何的删除操作都是将数据从数据库中彻底删除，所以运用时要特别小心。当要删除所有的数据时，程序会弹出删除确定的提示框，如图 7.23 所示。

该程序可从中国水利水电出版社网站下载。如果想直接运行，只要复制数据库文件 ResultDB.mdb 和程序的可执行文件 ResultManage.exe（不需要安装，可放于任何一个地方），并为数据库文件 ResultDB.mdb 创建名为 AccessDSN 的数据源后，双击 ResultManage.exe 文件即可运行程序。

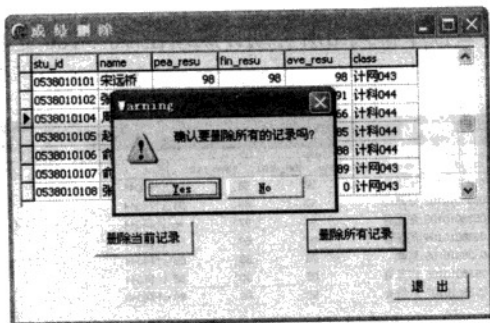


图 7.23 “成绩删除”子系统界面

7.6 小结

本章主要介绍基于 BDE 数据库开发的相关技术。介绍了常用的 BDE 组件及其相关的数据库组件。以此为基础，介绍了数据库浏览，查询设计，数据的插入、删除、修改等基本操作，最后以一个具体的应用为例详细地介绍了 BDE 数据库开发的一般步骤和方法。

通过对本章的学习，读者应掌握以下内容：

- 常用的 BDE 组件及其相关的数据库组件的基本属性和方法。
- 如何创建一个数据库连接，及如何打开和关闭数据库连接。
- 熟练掌握数据库的基本管理操作，包括数据浏览，查询设计，数据的插入、删除、修改等。
- 掌握 BDE 数据库开发的一般步骤和方法。

第 8 章 基于 ADO.NET 的数据库应用开发

当今, .NET 技术已经风靡全球, 得到了众多程序员的青睐, 已经在诸多领域中得到应用。作为 .NET 的核心技术之一, ADO.NET 集中了所有可以进行数据处理的类, 提供了平台互用和可收缩的数据访问功能。ADO .NET 由 Microsoft ActiveX Data Objects (ADO) 改进而来, 但它更多地显示了涵盖全面的数据库设计功能, 是对 ADO 的根本性改进, 成为 .NET 数据库应用程序的解决方案。本章主要介绍 ADO.NET 技术在数据库开发中的应用, 涉及内容主要包括:

- ADO.NET 技术的特点。
- 常用 ADO.NET 组件的属性和方法。
- 运用 ADO.NET 组件访问数据库。
- 使用 DataGrid 对象管理数据的方法。
- 开发 ADO.NET 数据库应用程序的基本方法。

8.1 关于 ADO.NET

8.1.1 ADO.NET 简介

ADO.NET 是在对 Microsoft ActiveX Data Objects (ADO) 改进的基础上发展而来的数据访问模型, 被认为是一个“跨时代的产品”。它经历了从 ODBC 到 ADO, 再由 ADO 到 ADO.NET 的发展过程, 是微软公司开发的一种用于 Windows.NET 开发的数据库访问接口。它提供了平台互用性和可伸缩的数据访问功能, 可以使用它来访问关系数据库系统 (如 SQL Server 2000、Oracle) 和其他许多具有 OLE DB 或 ODBC 提供程序的数据源。

ADO.NET 是专门为 .NET 框架而设计的 (是 .NET 框架中的核心技术), 是构建 .NET 数据库应用程序的基础。与 ADO 不同的是, ADO 使用 OLE DB 接口并基于微软的 COM 技术, 而 ADO.NET 拥有自己的 ADO.NET 接口并且基于微软的 .NET 体系架构。另外, ADO 是以记录集 (Recordset) 的形式存储数据, 而 ADO.NET 则以数据集 (DataSet) 的形式来存储。数据集是指数据库数据在内存中的复制副本, 一个数据集包含多个数据表, 这些表就组成了一个非连接的数据库数据视图。这种非连接的结构体系使得只有在读、写数据库时才需要使用数据库服务器资源, 因而提供了更好的可伸缩性。

ADO 是在线方式运作, 而 ADO.NET 则以离线方式运作。因此, 使用 ADO 连接数据库会占用较大的服务器系统资源, 而采用 ADO.NET 技术的应用程序则具有较高的系统性能。

ADO.NET 结构如图 8.1 所示。从该图中可以看出, ADO.NET 可分为 DataProvider 和 DataSet 两大部分, 所使用的数据源可以是数据库 (Database), 也可以是 XML 文件。DataProvider 的作用在于创建与数据源的连接并发出数据操作的命令。DataSet 则是用于在内存中定义数据的存储结构, 从逻辑意义上讲, DataSet 是多张数据表的集合, 并提供了多种对数据操作的方法。

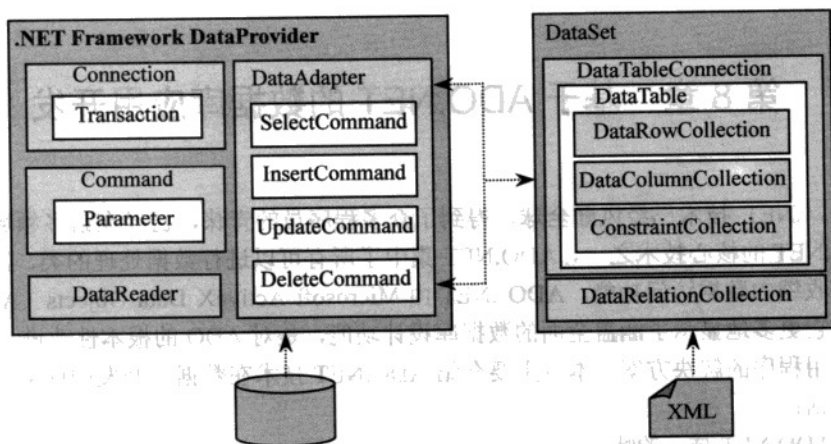



图 8.1 ADO.NET 结构

总之，ADO.NET 和 ADO 是两种不同的数据访问方式，利用 ADO.NET 技术开发数据库应用程序，可以有效地提高程序的整体性能和系统资源的利用率。ADO.NET 是 ADO 的升级版，是微软最新数据库访问技术的集成，它为使用者提供全新的数据访问机制，在提高数据访问效率的同时大大简化了程序员的工作量。

8.1.2 一个 ADO.NET 数据库应用程序

为使读者对 ADO.NET 数据库开发技术有一个感性的认识，本节通过一个简单的例子介绍在 Delphi 2005 中 ADO.NET 数据库应用程序开发的过程。具体创建步骤如下：

(1) 创建 Windows Forms Application（而不是 VCL Forms Application），命名为 ADOSample.dpr。创建的方法是：在 Delphi 2005 IDE 中选择菜单 File→New→Windows Forms Application 命令。

(2) 在窗体中添加 SqlConnection、SqlDataAdapter、DataSet、DataGrid 和 Button 组件各一个，均采用默认的名称。在窗体中选中的 SqlConnection 组件，然后在对象观察器中设置其ConnectionString 属性，方法如下：在对象观察器中找到 ConnectionString 属性并单击其右边的省略号按钮 ，将打开数据连接属性（Data Link Properties）对话框，如图 8.2 所示。

在数据连接属性对话框中，首先在第一个下拉列表框中选择相应的数据库服务器名称（本例设为本地机器名“MZQ”），然后选中“Use Windows NT Integrated security”单选框，最后在第二个下拉列表框中选择对应数据库服务器上的一个数据库（本例选择 SQL Server 自带的数据库 Northwind）。如果想确定连接是否成功，可以通过单击 Test Connection 按钮来测试创建的连接是否有效。如果有效，则单击“确定”按钮，数据连接创建成功。

通过以上操作后，SqlConnection 组件的 ConnectionString 属性值被设置为如下的字符串：
integrated security=SSPI;data source=MZQ;persist security info=False;initial catalog=Northwind

也就是说，上面的操作实际上是等价于执行下列的赋值语句：

```
SqlConnection1.ConnectionString := 'integrated security=SSPI;data source=MZQ;persist security info=False;
initial catalog=Northwind';
```

(3) 在窗体中选中的 SqlDataAdapter1 组件，然后在对象观察器中把 SelectCommand 属性

值设置为 `sqlSelectCommand1`, `SelectCommand.CommandText` 属性值设置为 “select * from Customers”, `SelectCommand.Connection` 属性值设置为 `SqlConnection1`, 如图 8.3 所示。

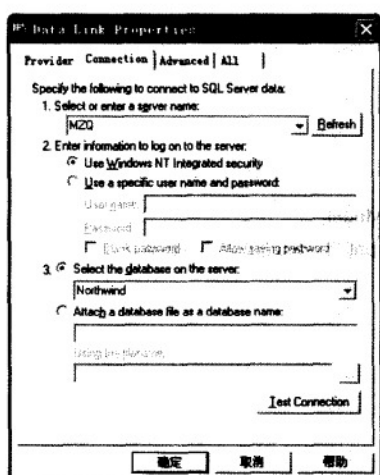


图 8.2 数据连接属性对话框

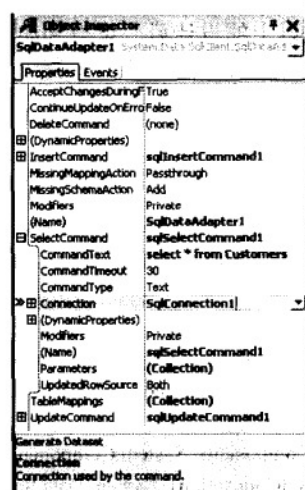


图 8.3 设置 SqlDataAdapter1 组件的相关属性

(4) 在窗体中选择 `DataSet` 组件, 然后在对象观察器中找到 `Tables` 属性项并单击其右边的省略号按钮 `...`, 打开数据表集编辑器 (`Tables Collection Editor`) 对话框。在该编辑器中单击 `Add` 按钮, 将增加一个 `DataTable` 对象, 接着在右边的方框中把 `(Name)` 属性值设置为 `Customers` (也可以设为其他的字符串), 如图 8.4 所示, 最后单击 `Close` 按钮。

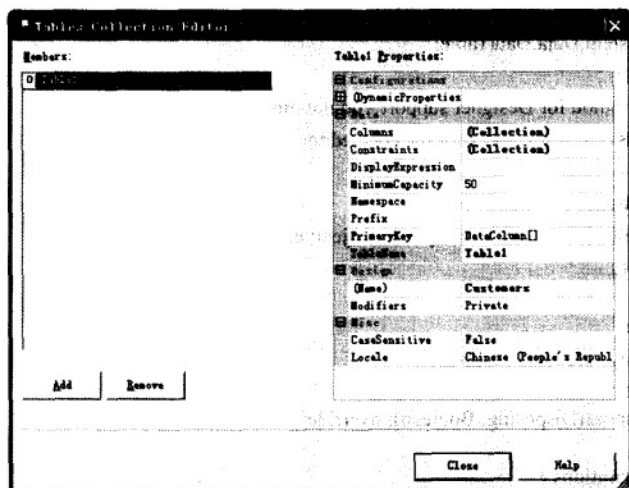


图 8.4 数据表集编辑器对话框

(5) 采用与上面类似的方法, 把 `DataGrid` 组件的 `DataSource` 属性值设置为 `Customers`。

(6) 把 `Button` 组件的 `Text` 属性值设置为 “打开数据库”, 然后在窗体中双击 “打开数据库” 按钮, 在出现的代码编辑器中编写事件的处理代码, 主要是在 `TWinForm.Button1_Click`

过程中添加下列代码:

```

SqlConnection1.Open;
SqlDataAdapter1.Fill(DataSet1.Tables[0]);
经过以上的设计后, 可得到如下的单元文件:
unit WinForm;

interface

uses
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data, System.Data.SqlClient, System.Globalization;

type
  TWinForm = class(System.Windows.Forms.Form)
  {$REGION 'Designer Managed Code'}
  strict private
    Components: System.ComponentModel.Container;
    SqlConnection1: System.Data.SqlClient.SqlConnection;
    sqlSelectCommand1: System.Data.SqlClient.SqlCommand;
    sqlInsertCommand1: System.Data.SqlClient.SqlCommand;
    sqlUpdateCommand1: System.Data.SqlClient.SqlCommand;
    sqlDeleteCommand1: System.Data.SqlClient.SqlCommand;
    SqlDataAdapter1: System.Data.SqlClient.SqlDataAdapter;
    DataSet1: System.Data.DataSet;
    DataGrid1: System.Windows.Forms.DataGrid;
    Button1: System.Windows.Forms.Button;
    Customer: System.Data.DataTable;
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
    procedure Button1_Click(sender: System.Object; e: System.EventArgs);
  {$ENDREGION}
  strict protected
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(Disposing: Boolean); override;
  private
    { Private Declarations }
  public
    constructor Create;
  end;

[assembly: RuntimeRequiredAttribute(typeof(TWinForm))]

```

implementation

```
{ $AUTOBOX ON }
```

```
{ $REGION 'Windows Form Designer generated code' }
```

```
/// <summary>
```

```
/// Required method for Designer support -- do not modify
```

```
/// the contents of this method with the code editor.
```

```
/// </summary>
```

```
procedure TWinForm.InitializeComponent;
```

```
type
```

```
TArrayOfSystem_Data_DataTable = array of System.Data.DataTable;
```

```
begin
```

```
Self.SqlConnection1 := System.Data.SqlClient.SqlConnection.Create;
```

```
Self.sqlSelectCommand1 := System.Data.SqlClient.SqlCommand.Create;
```

```
Self.sqlInsertCommand1 := System.Data.SqlClient.SqlCommand.Create;
```

```
Self.sqlUpdateCommand1 := System.Data.SqlClient.SqlCommand.Create;
```

```
Self.sqlDeleteCommand1 := System.Data.SqlClient.SqlCommand.Create;
```

```
Self.SqlDataAdapter1 := System.Data.SqlClient.SqlDataAdapter.Create;
```

```
Self.DataSet1 := System.Data.DataSet.Create;
```

```
Self.Customer := System.Data.DataTable.Create;
```

```
Self.DataGrid1 := System.Windows.Forms.DataGrid.Create;
```

```
Self.Button1 := System.Windows.Forms.Button.Create;
```

```
(System.ComponentModel.ISupportInitialize).BeginInit;
```

```
(System.ComponentModel.ISupportInitialize).BeginInit;
```

```
(System.ComponentModel.ISupportInitialize).BeginInit;
```

```
Self.SuspendLayout;
```

```
//
```

```
// SqlConnection1
```

```
//
```

```
Self.SqlConnection1.ConnectionString := 'integrated security=SSPI;data source=' +  
'rce=MZQ;persist security info=False;initial catalog=Northwind';
```

```
//
```

```
// sqlSelectCommand1
```

```
//
```

```
Self.sqlSelectCommand1.CommandText := 'Select * from Customers';
```

```
Self.sqlSelectCommand1.Connection := Self.SqlConnection1;
```

```
//
```

```
// SqlDataAdapter1
```

```
//
```

```
Self.SqlDataAdapter1.DeleteCommand := Self.sqlDeleteCommand1;
```

```
Self.SqlDataAdapter1.InsertCommand := Self.sqlInsertCommand1;
```

```
Self.SqlDataAdapter1.SelectCommand := Self.sqlSelectCommand1;
```

```
Self.SqlDataAdapter1.UpdateCommand := Self.sqlUpdateCommand1;
```

```
//
```

```
// DataSet1
```

```

//
Self.DataSet1.DataSetName := 'NewDataSet';
Self.DataSet1.Locale := System.Globalization.CultureInfo.Create('zh-CN');
Self.DataSet1.Tables.AddRange(TArrayOfSystem_Data_Table.Create(Self.Customer));
//
// Customer
//
Self.Customer.TableName := 'Table1';
//
// DataGrid1
//
Self.DataGrid1.DataMember := '';
Self.DataGrid1.DataSource := Self.Customer;
Self.DataGrid1.HeaderForeColor := System.Drawing.SystemColors.ControlText;
Self.DataGrid1.Location := System.Drawing.Point.Create(8, 8);
Self.DataGrid1.Name := 'DataGrid1';
Self.DataGrid1.Size := System.Drawing.Size.Create(472, 320);
Self.DataGrid1.TabIndex := 0;
//
// Button1
//
Self.Button1.Location := System.Drawing.Point.Create(216, 352);
Self.Button1.Name := 'Button1';
Self.Button1.TabIndex := 1;
Self.Button1.Text := '打开数据库';
Include(Self.Button1.Click, Self.Button1_Click);
//
// TWinForm
//
Self.AutoScaleBaseSize := System.Drawing.Size.Create(6, 14);
Self.ClientSize := System.Drawing.Size.Create(488, 398);
Self.Controls.Add(Self.Button1);
Self.Controls.Add(Self.DataGrid1);
Self.Name := 'TWinForm';
Self.Text := 'WinForm';
(System.ComponentModel.ISupportInitialize.Initialize(Self.DataSet1)).EndInit;
(System.ComponentModel.ISupportInitialize.Initialize(Self.Customer)).EndInit;
(System.ComponentModel.ISupportInitialize.Initialize(Self.DataGrid1)).EndInit;
Self.ResumeLayout(False);
end;
{$ENDREGION}

procedure TWinForm.Dispose(Disposing: Boolean);
begin
    if Disposing then
    begin

```

```

if Components <> nil then
    Components.Dispose();
end;
inherited Dispose(Disposing);
end;

constructor TWinForm.Create;
begin
    inherited Create;
    InitializeComponent; //初始化
end;

procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
begin
    SqlConnection1.Open; //激活连接
    SqlDataAdapter1.Fill(DataSet1.Tables[0]); //显示数据
end;

end.

```

(7) 运行程序, 在主界面中单击“打开数据库”按钮, 则数据库 NorthWind 被连接, 并且于 DBGrid 组件中显示数据表 Customers 中的数据, 如图 8.5 所示。

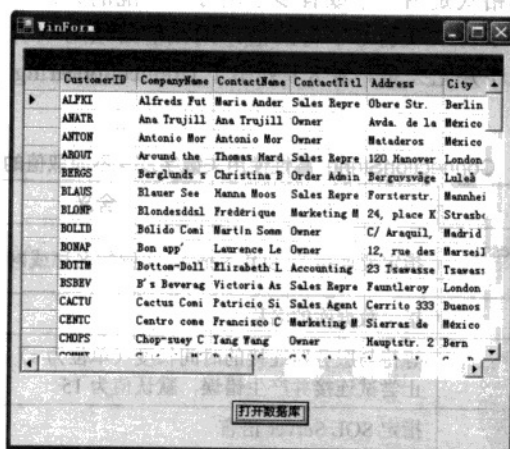


图 8.5 显示数据表 Customers

至此, 一个简单的基于 ADO.NET 技术的数据库应用程序的基本开发步骤已经全部介绍完毕。读者可以由此举一反三, 要不断拓展学习的深度和广度。

从这个例子也可以看出, 基于 ADO.NET 技术的数据库编程实际上是对 ADO.NET 组件的编程, 所以熟悉 ADO.NET 组件的属性和方法, 可以有效地提高 ADO.NET 数据库应用程序的开发水平。因此, 下一节着重介绍一些常用的 ADO.NET 组件。

8.2 熟悉常用的 ADO.NET 组件

对 ADO.NET 组件进行编程的目的就是从数据操作中分解出数据访问。为完成这一分解任务, ADO.NET 完全依赖于其两个核心组件, 即 .NET 框架数据提供者 (.NET Framework Data Provider) 和数据集 (DataSet), 如图 8.1 所示。前者主要包括 Connection、Command、DataReader 和 DataAdapter 等对象, 而后者则是包含一个或多个 DataTable 对象的集合, 这些对象是由数据行和数据列以及主键、外键等有关 DataTable 对象中数据之间关系的信息组成。可见, 使用 ADO.NET 组件实际上是使用它的对象, 所以下面通过介绍这些对象的使用方法来说明如何对 ADO.NET 组件进行编程。

8.2.1 SqlConnection 对象

在 ADO.NET 中, SqlConnection 对象用于连接 SQL Server 数据库。它包含的属性不多, 常用的是它的ConnectionString 属性。在程序 ADOSample.dpr 中, 该属性值被设置为下列字符串 (称为连接字符串):

```
'integrated security=SSPI;data source=MZQ;persist security info=False;initial catalog=Northwind'
```

可以看出, 它主要用于制定要连接的数据库、数据库服务器和一些安全信息等。连接字符串的基本格式为:

```
'关键字 1 = 值 1;[关键字 2 = 值 2; 关键字 3 = 值 3;...]'
```

即连接字符串的基本格式是由一个或者多个由分号分隔的关键字/值对组成的字符串。若要包括含有分号、单引号字符或双引号字符的值, 则该值必须用双引号括起来。关键字/值对在连接字符串中的先后顺序并无关系。表 8.1 列出了 ConnectionString 属性使用关键字及其不同取值的作用和意义。

表 8.1 ConnectionString 属性使用关键字及其不同取值的含义

关键字	含义
Data Source、Server、Address、Addr 或 Network Address	指定要连接的 SQL Server 实例的名称或网络地址
Initial Catalog 或 Database	指定数据库的名称
Connect Timeout 或 Connection Timeout	等待与服务器连接的时间长度 (单位为 s), 如果超过这个时间则终止尝试连接并产生错误, 默认值为 15
Current Language	指定 SQL Server 语言
Persist Security Info	该关键字的值为布尔型, 当其值取 false (no) 时, 如果连接是打开的或者一直处于打开状态, 那么密码等安全信息将不会作为连接的一部分返回。默认值为 false
Encrypt	该关键字的值为布尔型, 当其值取 true (或 yes) 时, 如果服务器端安装了证书, 则 SQL Server 将对所有在客户端和服务器之间传送的数据使用 SSL 加密, 默认值为 false (no)
Integrated Security 或 Trusted_Connection	该关键字的值为布尔型, 当其值取 false (no) 时, 必须在连接中指定用户 ID 和密码; 当值取 true (yes, sspi) 时, 则使用当前的 Windows 帐户凭据进行身份验证。默认值为 false

续表

关键字	含义
User ID	SQL Server 登录账户 (但建议改用 Integrated Security 或 Trusted_Connection 关键字)
Password 或 Pwd	SQL Server 账户登录的密码 (但建议改用 Integrated Security 或 Trusted_Connection 关键字)
Packet Size	指定与 SQL Server 实例进行通信的网络数据包的大小, 单位为 (B), 默认值为 8192
Workstation ID	连接到 SQL Server 的工作站的名称。默认值是本地计算机名称
Application Name	指定应用程序的名称, 默认值为: '.Net SqlClient Data Provider'
AttachDBFilename、extended properties 或 Initial File Name	指定连接数据库的主文件的名称 (含完整的路径名), 但其使用前提是由关键字 database 来指定数据库的名称
Network Library 或 Net	用于建立与 SQL Server 实例连接的网络数据库。其取值可为 dbnmptw (命名管道)、dbmsrpcn (多协议)、dbmsadsn (Apple Talk)、dbmsgnet (VIA)、dbmslpcn (共享内存) 及 dbmsspxn (IPX/SPX) 和 dbmsocn (TCP/IP)。相应的网络 DLL 必须安装在要连接的系统上。如果不指定网络而使用一个本地服务器, 如 "." 或 "(local)", 则使用共享内存。默认值为 'dbmsocn'

SqlConnection 对象还有一个经常用到的属性就是 State 属性, 通过访问该属性的值可以获取当前连接的状态。状态包括 Broken、Closed、Connecting、Executing、Fetching 及 Open。

SqlConnection 对象主要提供了以下几类方法:

- Open 方法。Open 方法用于打开已经设置好的数据库连接, 如果没有则它会创建一个新的 SQL Server 实例的连接。
- Close 方法。关闭与数据库的连接。
- StateChange 方法。当数据库连接状态发生改变时, 该方法被触发。
- InfoMessage 方法。数据库返回警告或者消息时, 该方法被触发。
- ChangeDatabase 方法。当改变连接的当前数据库时, 该方法被触发。

8.2.2 Command 对象

ADO.NET 的 Command 对象主要用于执行一些 SQL 语句, 包括用于返回数据、修改数据、运行存储过程及发送或检索参数信息的数据库命令。目前它有三种版本, 即 SqlCommand、OleDbCommand 和 OdbcCommand, 它们的属性和方法基本相同, 在此主要介绍 SqlCommand 对象。

SqlCommand 对象常用的属性和方法包括以下几种:

- CommandText 属性。定义命令的可执行文本, 包括 SQL 语句、存储过程等。
- CommandTimeout 属性。用于设置或返回终止执行命令之前需要等待的时间 (单位为秒), 默认值为 30。
- CommandType 属性。CommandType 属性用于决定 CommandText 属性值的格式。当 CommandType 属性值取 Text 时, CommandText 属性值为 SQL 语句; 当 CommandType 属性值取 StoredProcedure 时, 则 CommandText 属性值为存储过程; 当 CommandType

属性值取 TableDirect 时, 则 CommandText 属性值为要读取的表。

- Connection 属性。用于设置或返回数据库的连接对象。
- Parameters 属性。有些命令在执行时需要参数, Parameters 属性则用于传递给命令对象的参数。
- Cancel 方法。该方法用于取消命令的执行, 回到执行前的状态。
- CreateParameter 方法。该方法用于生成命令的参数。
- ExecuteNonQuery 方法。该方法用于执行 SQL 语句, 但这些语句没有返回结果集, 如 Insert、Update、Delete 等。例如, 下列代码可以实现对数据库 northwind 中表 Customers 的数据插入:

//创建数据库连接对象 myConnection, 当然在此之前要 SqlConnection 声明该对象

```
myConnection := SqlConnection.Create("Data Source = localhost;  
Integrated Security = SSPI;  
Initial Catalog = northwind");
```

//创建命令对象

```
myCommand := myConnection.CreateCommand;
```

//在命令对象中“装入”SQL 语句

```
myCommand.CommandText :=
```

```
'Insert into Customers(CustomerID, CompanyName) values("All", "Lenovo");'
```

//激活连接 myConnection

```
myConnection.Open;
```

//执行没有返回结果集的 SQL 语句

```
myCommand.ExecuteNonQuery;
```

- ExecuteReader 方法。执行有返回结果的 SQL 语句 (Select)、存储过程等, 返回结果存放在 DataReader 对象中。
- CreateXMLReader 方法。该方法与 ExecuteReader 方法类似, 不同的是其返回结果将以 XML 数据形式保存到 XMLReader 对象中。
- Prepare 方法。该方法用于在数据库实例上创建命令的一个准备 (或编译) 命令版本。
- ResetCommandTimeout 方法。用于恢复 CommandTimeout 属性的默认值。

8.2.3 DataReader 对象

对于 ADO 开发者来说, DataReader 是一个新对象, 其最大的优点就是执行效率高, 在体积和开销上它比数据集小, 属于高度专用型对象, 占用内存小。它是服务端的游标, 在读取数据时它与服务器的连接始终是打开的。但是其缺点也很明显, 它只能以单项向前的次序访问记录, 所以仅用于数据浏览等功能非常单一的设计中。即 DataReader 对象可从数据源中提供高性能的数据流, 但只能检索只读的数据流, 不能写入, 并且只能从头至尾往下读, 而不能只读某行数据。

DataReader 对象常用的属性和方法包括以下种:

- FieldCount。返回字段的数目。
- IsClosed。返回 DataReader 对象是否关闭的状态, 如果关闭则返回 true, 否则返回 false。
- RecordsAffected。返回执行 insert、delete 或 update 后受到影响的行数。
- Close()方法。用于关闭 DataReader 对象。

- **GetDataTypeName(n)**方法。返回第 $n+1$ 列的源数据类型名称。
- **GetFieldType(n)**方法。返回第 $n+1$ 列的数据类型
- **GetName(n)**方法。返回 $n+1$ 列的字段名称
- **GetOrdinal(name)**方法。返回字段名称为 name 的字段列号。
- **GetValue(n)**方法。返回 $n+1$ 列的内容。
- **GetValues(arrays)**方法。返回所有字段的内容，并将内容放在 arrays 数组中，数组大小与字段数目相等。
- **IsDBNull(n)**方法。用于判断第 $n+1$ 列是否为 Null，为 Null 则返回 true，否则返回 false。
- **Read()**方法。把记录指针向下一行移动，如果下一行没有了（当前行是最后一行），则返回 false，否则返回 true。使用该方法可以从查询结果中读取数据。

需要注意的是，由于 **DataReader** 对象与服务器的连接在读取数据时始终打开，所以每次使用 **DataReader** 对象后要及时调用 **Close** 方法关闭它。

8.2.4 一个应用实例

上面介绍了 ADO.NET 的三个重要的对象，下面将运用这些对象来完成在数据库 northwind 中对数据表 Customers 的插入和浏览操作。具体步骤如下：

- (1) 在 Delphi 2005 IDE 中创建一个 Windows Form Application，命名为 **ADOTest.dpr**。
- (2) 在窗体中加入一个 **ListBox** 组件，**TextBox**、**Label**、**Button** 和 **GroupBox** 组件各两个，适当调整各组件的位置和大小，结果如图 8.6 所示。

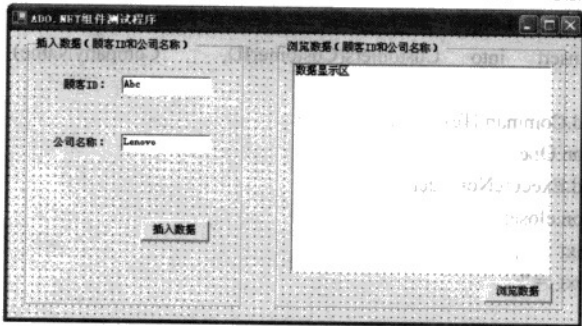


图 8.6 ADOTest.dpr 程序界面

- (3) 对各组件，设置其必要的属性，结果如表 8.2 所示。

表 8.2 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
ListBox	ListBox1	Items	数据显示区
TextBox	TextBox1	Text	Abc
	TextBox2	Text	Lenovo
Label	Label1	Text	顾客 ID:
	Label2	Text	公司名称:

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Button	Button1	Text	插入数据
	Button2	Text	浏览数据
GroupBox	GroupBox1	Text	插入数据 (顾客 ID 和公司名称)
	GroupBox2	Text	浏览数据 (顾客 ID 和公司名称)
Form	Form1	Caption	ADO.NET 组件测试程序

(4) 在窗体设计器中双击“插入数据”按钮, 进入代码编辑器, 编写该按钮所对应的事件处理过程。该过程代码如下:

```

procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var myConnection:SqlConnection;
    myCommand:SqlCommand;
    myReader: SqlDataReader;
    name,stu_id,str: string;

begin
    myConnection := SqlConnection.Create('Data Source = localhost; Integrated Security = SSPI; Initial
Catalog = northwind');
    myCommand := myConnection.CreateCommand;
    name := TextBox1.Text;
    stu_id := TextBox2.Text;
    str := 'Insert into Customers(CustomerID,      CompanyName) values(''+name.trim+'',
''+stu_id.trim+'')';
    myCommand.CommandText := str;
    myConnection.Open;
    myCommand.ExecuteNonQuery;
    myConnection.close;
    TextBox1.Text := '';
    TextBox2.Text := '';
end;

```

(5) 采用与 (4) 类似的方法进入代码编辑器, 编写“浏览数据”按钮所对应的事件处理过程。该过程代码如下:

```

procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);

var myConnection:SqlConnection;
    myCommand:SqlCommand;
    myReader: SqlDataReader;
    str: string;

begin
    myConnection := SqlConnection.Create('Data Source = localhost; Integrated Security = SSPI; Initial
Catalog = northwind');

```

```

myCommand := myConnection.CreateCommand;
myCommand.CommandText := 'select * from Customers';
myConnection.Open;
myReader := MyCommand.ExecuteReader;
ListBox1.Items.Clear;
while myReader.Read do
begin
    str := myReader.GetString(0);
    str := str + ', ' + myReader.GetString(1);
    ListBox1.Items.Add(str);
end;
myConnection.close;

```

end;

经过上述的窗体设计和代码设计后, 得到单元文件 WinForm 的代码如下:

```
unit WinForm;
```

```
interface
```

```
uses
```

```

    System.Drawing, System.Collections, System.ComponentModel,
    System.Windows.Forms, System.Data, System.Data.SqlClient;

```

```
type
```

```

TWinForm = class(System.Windows.Forms.Form)
{$REGION 'Designer Managed Code'}
strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
    Components: System.ComponentModel.Container;
    ListBox1: System.Windows.Forms.ListBox;
    TextBox1: System.Windows.Forms.TextBox;
    TextBox2: System.Windows.Forms.TextBox;
    Label1: System.Windows.Forms.Label;
    Label2: System.Windows.Forms.Label;
    Button1: System.Windows.Forms.Button;
    Button2: System.Windows.Forms.Button;
    GroupBox1: System.Windows.Forms.GroupBox;
    GroupBox2: System.Windows.Forms.GroupBox;
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
    procedure TWinForm_Load(sender: System.Object; e: System.EventArgs);
    procedure Button2_Click(sender: System.Object; e: System.EventArgs);

```

```

    procedure Button1_Click(sender: System.Object; e: System.EventArgs);
{$ENDREGION}
strict protected
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(Disposing: Boolean); override;
private
    { Private Declarations }
public
    constructor Create;
end;
[assembly: RuntimeRequiredAttribute(TypeOf(TWinForm))]
implementation

{$AUTOBOX ON}

{$REGION 'Windows Form Designer generated code'}
/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWinForm.InitializeComponent;
type
    TArrayOfSystem_Object = array of System.Object;
begin
    Self.ListBox1 := System.Windows.Forms.ListBox.Create;
    Self.TextBox1 := System.Windows.Forms.TextBox.Create;
    Self.TextBox2 := System.Windows.Forms.TextBox.Create;
    Self.Label1 := System.Windows.Forms.Label.Create;
    Self.Label2 := System.Windows.Forms.Label.Create;
    Self.Button1 := System.Windows.Forms.Button.Create;
    Self.Button2 := System.Windows.Forms.Button.Create;
    Self.GroupBox1 := System.Windows.Forms.GroupBox.Create;
    Self.GroupBox2 := System.Windows.Forms.GroupBox.Create;
    Self.GroupBox1.SuspendLayout;
    Self.GroupBox2.SuspendLayout;
    Self.SuspendLayout;
    //
    // ListBox1
    //
    Self.ListBox1.ItemHeight := 12;
    Self.ListBox1.Items.AddRange(TArrayOfSystem_Object.Create('数据显示区'));
    Self.ListBox1.Location := System.Drawing.Point.Create(16, 24);
    Self.ListBox1.Name := 'ListBox1';
    Self.ListBox1.Size := System.Drawing.Size.Create(288, 232);

```

```
Self.ListBox1.TabIndex := 0;
//
// TextBox1
//
Self.TextBox1.Location := System.Drawing.Point.Create(104, 40);
Self.TextBox1.Name := 'TextBox1';
Self.TextBox1.TabIndex := 1;
Self.TextBox1.Text := 'Abc';
//
// TextBox2
//
Self.TextBox2.Location := System.Drawing.Point.Create(104, 104);
Self.TextBox2.Name := 'TextBox2';
Self.TextBox2.TabIndex := 2;
Self.TextBox2.Text := 'Lenovo';
//
// Label1
//
Self.Label1.Location := System.Drawing.Point.Create(39, 45);
Self.Label1.Name := 'Label1';
Self.Label1.Size := System.Drawing.Size.Create(60, 23);
Self.Label1.TabIndex := 3;
Self.Label1.Text := '顾客 ID: ';
//
// Label2
//
Self.Label2.Location := System.Drawing.Point.Create(28, 109);
Self.Label2.Name := 'Label2';
Self.Label2.Size := System.Drawing.Size.Create(72, 23);
Self.Label2.TabIndex := 4;
Self.Label2.Text := '公司名称: ';
//
// Button1
//
Self.Button1.Location := System.Drawing.Point.Create(128, 200);
Self.Button1.Name := 'Button1';
Self.Button1.TabIndex := 5;
Self.Button1.Text := '插入数据';
Include(Self.Button1.Click, Self.Button1_Click);
//
// Button2
//
Self.Button2.Location := System.Drawing.Point.Create(232, 264);
Self.Button2.Name := 'Button2';
Self.Button2.TabIndex := 6;
Self.Button2.Text := '浏览数据';
```



```

Include(Self.Button2.Click, Self.Button2_Click);
//
// GroupBox1
//
Self.GroupBox1.Controls.Add(Self.TextBox1);
Self.GroupBox1.Controls.Add(Self.TextBox2);
Self.GroupBox1.Controls.Add(Self.Button1);
Self.GroupBox1.Controls.Add(Self.Label1);
Self.GroupBox1.Controls.Add(Self.Label2);
Self.GroupBox1.Location := System.Drawing.Point.Create(19, 8);
Self.GroupBox1.Name := 'GroupBox1';
Self.GroupBox1.Size := System.Drawing.Size.Create(240, 296);
Self.GroupBox1.TabIndex := 9;
Self.GroupBox1.TabStop := False;
Self.GroupBox1.Text := '插入数据（顾客 ID 和公司名称）';
//
// GroupBox2
//
Self.GroupBox2.Controls.Add(Self.ListBox1);
Self.GroupBox2.Controls.Add(Self.Button2);
Self.GroupBox2.Location := System.Drawing.Point.Create(296, 8);
Self.GroupBox2.Name := 'GroupBox2';
Self.GroupBox2.Size := System.Drawing.Size.Create(320, 296);
Self.GroupBox2.TabIndex := 10;
Self.GroupBox2.TabStop := False;
Self.GroupBox2.Text := '浏览数据（顾客 ID 和公司名称）';
//
// TWinForm
//
Self.AutoScaleBaseSize := System.Drawing.Size.Create(6, 14);
Self.ClientSize := System.Drawing.Size.Create(632, 318);
Self.Controls.Add(Self.GroupBox2);
Self.Controls.Add(Self.GroupBox1);
Self.Name := 'TWinForm';
Self.Text := 'ADO.NET 组件测试程序';
Include(Self.Load, Self.TWinForm_Load);
Self.GroupBox1.ResumeLayout(False);
Self.GroupBox2.ResumeLayout(False);
Self.ResumeLayout(False);
end;
{$ENDREGION}

procedure TWinForm.Dispose(Disposing: Boolean);
begin
    if Disposing then
        begin

```

```

    if Components <> nil then
        Components.Dispose();
    end;
    inherited Dispose(Disposing);
end;
constructor TWinForm.Create;
begin
    inherited Create;
    // Required for Windows Form Designer support
    InitializeComponent;
    // TODO: Add any constructor code after InitializeComponent call
end;

procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var myConnection:SqlConnection;
    myCommand:SqlCommand;
    myReader: SqlDataReader;
    name,stu_id,str: string;
begin
    myConnection := SqlConnection.Create('DataSource = localhost;
                                         Integrated Security = SSPI;
                                         Initial Catalog = northwind');

    myCommand := myConnection.CreateCommand;
    name := TextBox1.Text;
    stu_id := TextBox2.Text;
    str :=
'Insert into Customers(CustomerID, CompanyName) values('"+name.trim+"', '"+stu_id.trim+"')';
    myCommand.CommandText := str;
    myConnection.Open;
    myCommand.ExecuteNonQuery;
    myConnection.close;
    TextBox1.Text := '';
    TextBox2.Text := '';
end;

procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);
var myConnection:SqlConnection;
    myCommand:SqlCommand;
    myReader: SqlDataReader;
    str: string;
begin
    myConnection := SqlConnection.Create('DataSource = localhost;
                                         Integrated Security = SSPI;
                                         Initial Catalog = northwind');

    myCommand := myConnection.CreateCommand;

```

```
myCommand.CommandText := 'select * from Customers';
myConnection.Open;
myReader := MyCommand.ExecuteReader;
ListBox1.Items.Clear;
while myReader.Read do
begin
    str := myReader.GetString(0);
    str := str + ', ' + myReader.GetString(1);
    ListBox1.Items.Add(str);
end;
myConnection.close;
end;
end.
```

(6) 编译、运行 ADOTest 程序，在运行界面中单击“插入数据”按钮（采用两个文本框的默认值作为插入值，分别为 Abc 和 Lenovo）；然后单击“浏览数据”按钮，发现两个数据都已经被插入到数据表 Customers 中，如图 8.7 所示。

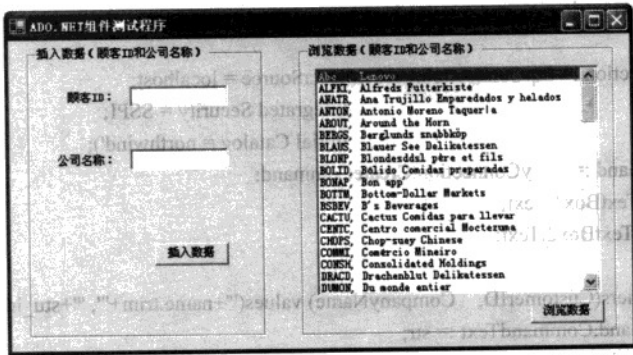


图 8.7 ADOTest.dpr 程序的运行界面

8.2.5 DataSet 对象

DataSet 对象是 ADO.NET 组件中的核心成员之一，它对于实现基于 ADO.NET 的数据操作起着关键作用。当从物理数据库中读取数据以后，DataSet 就成为这些数据在内存中的临时“数据容器”，同时它在客户端实现读取、更新数据库等过程中起到了中间部件的作用。因此，不管是数据写入还是读出，DataSet 都发挥着桥梁的作用，其作用可以用图 8.8 示意。



图 8.8 DataSet 对象作用示意

DataReader 对象也有类似的作用，但是 DataReader 对象只能用于读取数据库的数据，却

不能写入数据；而 DataSet 对象则既可读也可写，这是两者的不同之处。

DataReader 对象可以用于多个不同的数据源，用于 XML 数据，或用于管理应用程序本地的数据，但不管对什么样的数据源，都可以为用户提供一致的关系编程模型。图 8.9 显示了 DataSet 对象的结构模型，它位于 System.Data 名字空间中。

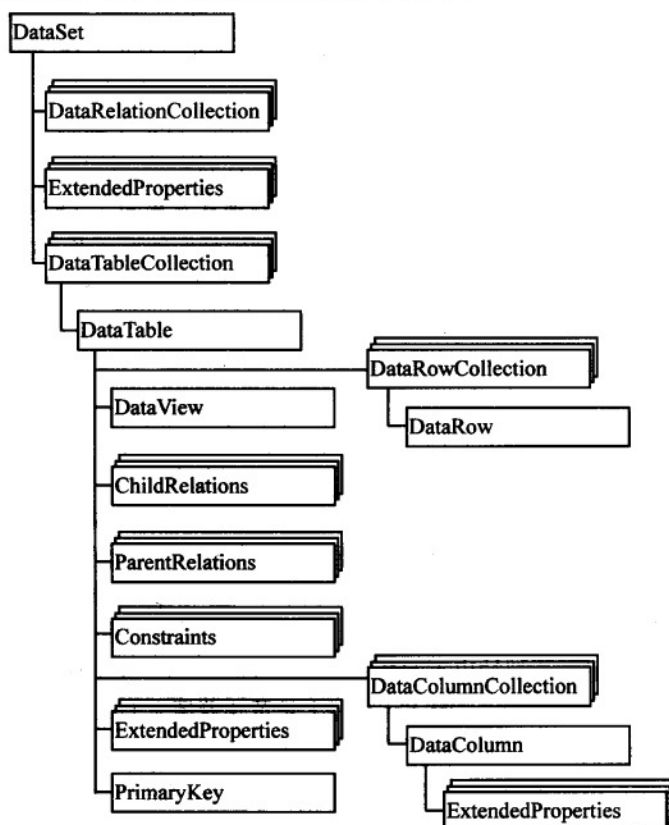


图 8.9 DataSet 对象结构模型

从图 8.9 可以看出，DataSet 对象结构比较复杂，在 DataSet 对象的下一层中是 DataTableCollection 对象、DataRelationCollection 对象和 ExtendedProperties 对象。DataTableCollection 对象是 DataSet 中多个 DataTable 对象的管理者，而两个 DataTable 对象之间的父/子关系形成了 DataRelation 对象，所有的 DataRelation 对象都由 DataRelationCollection 对象来管理。ExtendedProperties 则是一个属性集（PropertyCollection），用以存放各种自定义数据，如生成数据集的 SELECT 语句等。

DataSet 的使用方法一般有三种：

- (1) 通过数据提供者（DataAdapter）用数据库中的数据表填充 DataSet 对象。
- (2) 利用代码编程方法创建 DataTables、DataRelations 和 Constraints 对象，然后用这些对象填充 DataSet。
- (3) 用 XML 数据流或文本填充到 DataSet。

方法(1)用得较多,是本章介绍的主要内容。

掌握 DataSet 使用方法还需要涉及 ADO.NET 的另外一个核心常用成员——DataAdapter 对象(数据适配器对象)。DataSet 是独立于数据源的,而它又可以实现从不同数据源中抽取数据。实际上对于每一种数据源,如果 DataSet 要从中抽取数据,就必须有一个数据提供者(Data Provider),而这种提供者正是包含了 DataAdapter 对象。也就是说,数据提供者主要是通过 DataAdapter 对象来实现数据从数据库到 DataSet 对象的填充,或者从 DataSet 对象到数据库的回填。

不同的数据提供者包含的 DataAdapter 对象也不一样。SQL Server 数据提供者包含的是 SqlDataAdapter 对象,OLE DB 数据提供者包含 OleDbDataAdapter 对象,而 ODBC 数据提供者包含的是 OdbcDataAdapter 对象。这些对象的工作原理基本相同,它们都是使用 Connection 对象连接到数据源,然后使用 Command 对象从数据源中抽取数据或者回填数据。

DataSet 对象要与 DataAdapter 对象结合起来使用才能完成对数据库的访问。因此,需要进一步介绍 DataAdapter 对象常用的属性和方法。这些属性和方法主要包括:

(1) SelectCommand 属性。SelectCommand 属性实际上是一个 Command 对象(下面讲到的 InsertCommand 属性、UpdateCommand 属性和 DeleteCommand 属性等也都是 Command 对象),用于从数据源中抽取数据。

(2) InsertCommand 属性。用于把数据从 DataSet 对象中插入到数据源(数据库)中。

(3) UpdateCommand 属性。用于更新和修改数据源中的数据,数据是来自 DataSet 对象。

(4) DeleteCommand 属性。用于删除数据源中满足既定条件的数据。

(5) Fill()方法。Fill()方法用于把 SelectCommand 返回的数据填充到 DataSet。填充过程分为两步:首先通过 DataAdapter 的 SelectCommand 属性从数据库中抽出需要的数据,然后再利用 DataAdapter 的 Fill()方法把抽出来的数据填充 DataSet。上文已经指出,DataSet 实际上多个 DataTable 对象的集合,所以填充 DataSet 实际上是意味着填充 DataSet 中的 DataTable 对象。因此,在调用 Fill()方法时要以 DataSet 对象名为参数。

下面这段代码中,创建了 SqlConnection、SqlCommand、SqlDataAdapter 和 DataSet 实例,并通过它们连接到 SQL Server 中自带的数据库 pubs,把其中的数据表 authors 填充到 DataSet 的 DataTable 对象中。

```
var myConnection: SqlConnection;  
    myCommand: SqlCommand;  
    myDataAdapter: SqlDataAdapter;  
    myDataSet: DataSet;  
  
begin  
    //实例化 SqlConnection 对象  
    myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security  
info=False;initial catalog= pubs');  
    //实例化 SqlCommand 对象  
    myCommand := SqlCommand.Create('select * from authors');  
    //实例化 SqlDataAdapter 对象,并初始化  
    myDataAdapter := SqlDataAdapter.Create;  
    myDataAdapter.SelectCommand := myCommand;
```

```
myDataAdapter.SelectCommand.Connection := myConnection;  
//激活数据库连接  
myConnection.Open;  
//实例化 DataSet 对象  
myDataSet := DataSet.Create;  
//把表 authors 中的数据填充到 DataSet 的 DataTable 对象中  
myDataAdapter.Fill(myDataSet);  
myConnection.close;  
end;
```

8.3 使用 DataGrid 对象浏览数据

在 Windows Form 应用程序中,有很多方法可以浏览数据(如利用 DataReader),但最直观、最常用的是利用 .NET 框架下的 DataGrid 对象来浏览数据。DataGrid 对象是按照行列的方式显示数据,可以设置不同的显示风格,它具有丰富而强大的数据表格显示能力。

如果是单个数据表的数据源,只要把 DataGrid 绑定到该数据源,数据即可自动填充 DataGrid 对象。方法是在设计时设置 DataGrid 对象的 DataSource 属性值,数据源记录集中的数据就会自动填充该对象,同时自动设置该控件的列标头。在数据显示时,还可以编辑该网格的列,删除、重新安排、添加列标头,或者调整任意一列的宽度等。在运行时,可以动态切换 DataSource 属性值来显示不同的数据表,或者可以修改当前数据库的查询,以返回一个不同的记录集。

例如,如果有若干个 ADO.NET 对象,每个对象连接不同的数据源,可以简单地将 DataSource 从一个 ADO.NET 对象重新设置为另一个 ADO.NET 对象:

```
DataGrid1.DataSource := DataTable2;
```

此外,DataGrid 对象常用的方法和事件包括:

- Refresh 方法。Refresh 方法用于刷新 DataGrid 对象,以使得变动的数据能实时显示出来。
- Click 事件。当单击 DataGrid 对象时将触发 Click 事件。需要注意的是,单击某一个单元时并不触发该事件,而需要单击单元的“边框”才会触发该事件。
- DataSourceChanged 事件。当 DataGrid 对象的 DataSource 属性值被改变时,该事件被触发。

在前面介绍的例子(ADOSample.dpr)中,运用了 DataGrid 对象来显示数据。读者可能已经注意到了,虽然只是实现了一个简单的数据显示功能,程序的代码(自动产生)却比较长。

在本节中,可以运用本章前面学到的有关对象的知识,通过自己编程的方法来完成程序 ADOSample.dpr 一样的功能。可以按照以下几个步骤来完成:

(1) 创建一个 Windows Forms Application,命名为 ADOSample2.dpr(以示与 ADOSample.dpr 的区别)。创建的方法是:在 Delphi 2005 IDE 中选择菜单 File→New→Windows Forms Application 命令。

(2) 在窗体中仅添加两个组件,即 DataGrid 和 Button 组件,并把 Button 组件的 Text 属性值设置为“打开数据库连接”,组件的其他属性值采用默认值,如图 8.10 所示。

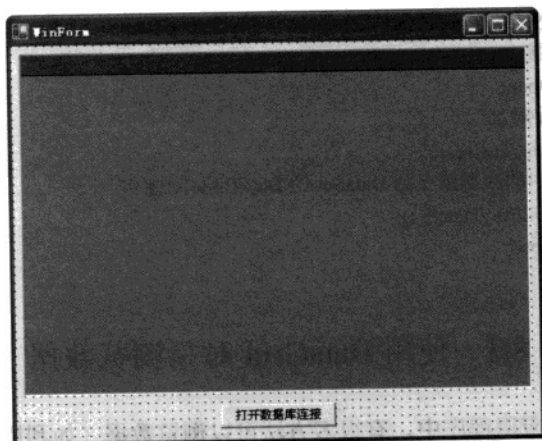


图 8.10 程序 ADOSample2.dpr 的设计界面

(3) 在窗体设计中双击“打开数据库连接”按钮，进入代码编辑器，然后在 uses 部分加上名字空间 System.Data.SqlClient，因为 SqlConnection 等对象位于该空间中。

(4) 首先声明 SqlConnection、SqlDataAdapter、DataSet 等对象，在 TWinForm.Button1_Click 函数中完成这一步。

```
var myConnection: SqlConnection;
    myCommand: SqlCommand;
    myDataAdapter: SqlDataAdapter;
    myDataSet: DataSet;
```

(5) 用构造函数实例化连接 SQL Server 数据库的连接对象 myConnection，使之连接到本机上的 Northwind 数据库实例。如果 SQL Server 已经安装在其他机器上，则使关键字 data source 等于相应的机器名称即可。

```
myConnection := SqlConnection.Create('integrated security=SSPI;
                                     data source=localhost;
                                     persist security info=False;
                                     initial catalog=Northwind');
```

(6) 用构造函数实例化 SqlCommand 对象，使之载入“select * from customers”这一 SQL 命令：

```
myCommand := SqlCommand.Create('select * from customers');
```

(7) 实例化 myDataAdapter 对象，并用 myCommand 和 myConnection 初始化：

```
myDataAdapter := SqlDataAdapter.Create;
myDataAdapter.SelectCommand := myCommand;
myDataAdapter.SelectCommand.Connection := myConnection;
```

(8) 激活连接 myConnection：

```
myConnection.Open;
```

(9) 实例化 myDataSet 对象：

```
myDataSet := DataSet.Create;
```

(10) 用 myDataAdapter 对象从数据源中抽取数据并填充到 myDataSet 对象，这一操作由 myDataAdapter 对象提供的 Fill() 方法来完成：

```
myDataAdapter.Fill(myDataSet);
```

(11) 把组件 DataGrid1 的 DataSource 属性设置为已被填充的 myDataSet 中的 DataTable 对象——myDataSet.Tables[0]。myDataSet 是若干个 DataTable 对象的集合, 由于 myDataSet 只进行了一次填充, 所以, 该被填充的 DataTable 对象是集合中的第一个元素, 表示为 myDataSet.Tables[0]。

```
DataGrid1.DataSource := myDataSet.Tables[0];
```

也可以用 DataTable 对象名称来访问 DataSet 中的某一个 DataTable 对象, 但这样做的前提是在填充 DataSet 时必须命名生成 DataTable 对象。于是, 步骤 (10) 和 (11) 对应的语句分别改为:

```
myDataAdapter.Fill(myDataSet, 'stuTable');
```

和

```
DataGrid1.DataSource := myDataSet.Tables.Item['stuTable'];
```

其中, 'stuTable' 是任意合法的字符串, 表示生成的 DataTable 对象的名称。

(12) 关闭数据库连接:

```
myConnection.close;
```

经过上述 (4) ~ (12) 步的代码编写后, 就完成了程序 ADOSample2 既定的浏览功能。可以发现, 如果熟悉 ADO.NET 对象, 则能很容易地编写出应用程序, 而不必像程序 ADOSample 那样要经过烦琐的过程, 而且程序代码也很短, 可读性强。程序 ADOSample2 的单元文件代码如下:

```
unit WinForm;

interface

uses
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data, System.Data.SqlClient;

type
  TWinForm = class(System.Windows.Forms.Form)
  {$REGION 'Designer Managed Code'}
  strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
    Components: System.ComponentModel.Container;
    DataGrid1: System.Windows.Forms.DataGrid;
    Button1: System.Windows.Forms.Button;
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
    procedure Button1_Click(sender: System.Object; e: System.EventArgs);
  {$ENDREGION}
```



```

strict protected
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  procedure Dispose(Disposing: Boolean); override;
private
  { Private Declarations }
public
  constructor Create;
end;

[assembly: RuntimeRequiredAttribute(TypeOf(TWinForm))]

implementation

{$AUTOBOX ON}

{$REGION 'Windows Form Designer generated code'}
/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWinForm.InitializeComponent;
begin
  Self.DataGrid1 := System.Windows.Forms.DataGrid.Create;
  Self.Button1 := System.Windows.Forms.Button.Create;
  (System.ComponentModel.ISupportInitialize(Self.DataGrid1)).BeginInit;
  Self.SuspendLayout;
  //
  // DataGrid1
  //
  Self.DataGrid1.DataMember := '';
  Self.DataGrid1.HeaderForeColor := System.Drawing.SystemColors.ControlText;
  Self.DataGrid1.Location := System.Drawing.Point.Create(8, 8);
  Self.DataGrid1.Name := 'DataGrid1';
  Self.DataGrid1.Size := System.Drawing.Size.Create(496, 336);
  Self.DataGrid1.TabIndex := 0;
  //
  // Button1
  //
  Self.Button1.Location := System.Drawing.Point.Create(200, 355);
  Self.Button1.Name := 'Button1';
  Self.Button1.Size := System.Drawing.Size.Create(112, 24);
  Self.Button1.TabIndex := 1;
  Self.Button1.Text := '打开数据库连接';
  Include(Self.Button1.Click, Self.Button1_Click);

```

```
// TWinForm
Self.AutoScaleBaseSize := System.Drawing.Size.Create(6, 14);
Self.ClientSize := System.Drawing.Size.Create(512, 390);
Self.Controls.Add(Self.Button1);
Self.Controls.Add(Self.DataGrid1);
Self.Name := 'TWinForm';
Self.Text := 'WinForm';
(System.ComponentModel.ISupportInitialize.SupportInitialize(Self.DataGrid1)).EndInit;
Self.ResumeLayout(False);
end;
{$ENDREGION}

procedure TWinForm.Dispose(Disposing: Boolean);
begin
    if Disposing then
        begin
            if Components <> nil then
                Components.Dispose();
            end;
            inherited Dispose(Disposing);
        end;
end;

constructor TWinForm.Create;
begin
    inherited Create;
    // Required for Windows Form Designer support
    InitializeComponent;
    // TODO: Add any constructor code after InitializeComponent call
end;

procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var myConnection: SqlConnection;
    myCommand: SqlCommand;
    myDataAdapter: SqlDataAdapter;
    myDataSet: DataSet;
begin
    myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security
info=False;initial catalog=Northwind');
    myCommand := SqlCommand.Create('select * from customers');
    myDataAdapter := SqlDataAdapter.Create;
    myDataAdapter.SelectCommand := myCommand;
    myDataAdapter.SelectCommand.Connection := myConnection;
    myConnection.Open;
```

```
myDataSet := DataSet.Create;
myDataAdapter.Fill(myDataSet);
DataGrid1.DataSource := myDataSet.Tables[0];
myConnection.close;

end;

end.
```

(13) 编译、运行 ADOSample2，在运行界面中单击“打开数据库连接”按钮，数据库实例 Northwind 中的数据表 customers 被显示到 DataGrid 对象中，如图 8.11 所示。

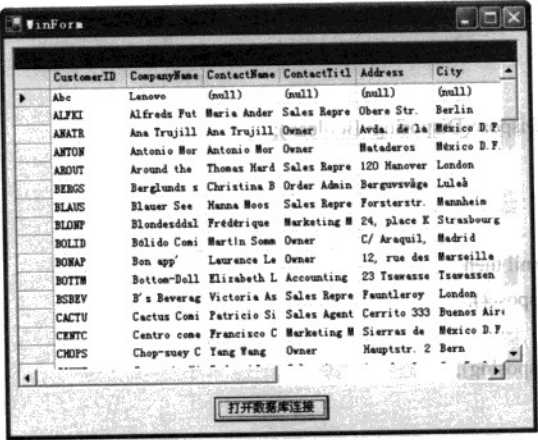


图 8.11 ADOSample2 的运行界面

8.4 ADO.NET 数据库开发实例

下面通过一个实例来说明基于 ADO.NET 组件的数据库开发方法。

数据的添加、修改和删除操作对应的 SQL 语句分别是 insert、update 和 delete，它们有一个共同的特点就是执行后没有返回值。从前面 ADO.NET 组件的介绍可知，可以调用 Command 对象的 ExecuteNonQuery 方法来执行这些语句。所以，一个自然的想法就是使用 Command 对象来实现数据的添加、修改和删除功能。数据显示则在 DataGrid 组件中完成，但也是通过 Command 对象由 DataAdapter 的填充方法 Fill 来实现。

该实例连接的是 6.3.2 节和 6.3.3 节介绍和创建的 MyDatabase 数据库和 stu 数据表。

8.4.1 实例的设计与数据显示

创建名为 ADOInsUpdDel 的工程，然后在窗体中放入 TabControl 组件，并把该组件的 Dock 属性值设置为 Fill，找到并单击 TapPages 属性项右边的省略号按钮，打开标签页编辑器。

在该编辑器中依次创建四个标签页，其 Text 属性值分别设置为“显示数据”、“添加数据”、“修改数据”、“删除数据”。然后在“显示数据”标签页上放入 DataGrid 对象（用于显示数据）和 Button 组件，并把 Button 组件的 Text 属性值设置为“显示”，设计结果如图 8.12 所示。

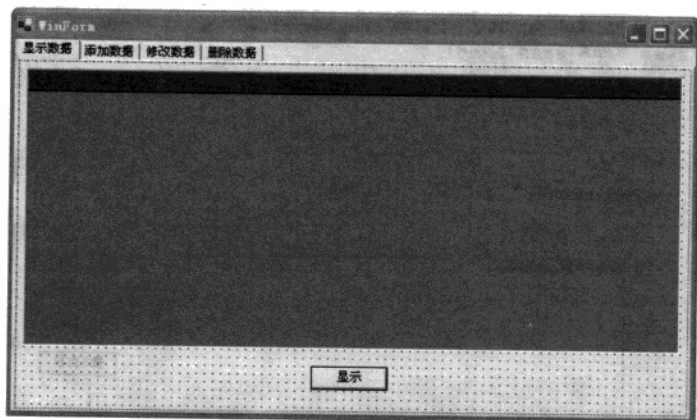


图 8.12 “显示数据”标签页

在“显示”按钮的 OnClick 事件处理过程 TWinForm.Button1_Click 中添加相应的代码，从而能在 DataGrid 对象显示数据。修改后的过程 TWinForm.Button1_Click 代码如下：

```
procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var
  myConnection: SqlConnection;
  myCommand: SqlCommand;
  myDataAdapter: SqlDataAdapter;
  myDataSet: DataSet;

begin
  myConnection := SqlConnection.Create('integrated security=SSPI;
                                     data source=localhost;
                                     persist security info=False;
                                     initial catalog=MyDatabase');

  myCommand := SqlCommand.Create('select * from stu');
  myDataAdapter := SqlDataAdapter.Create;
  myDataAdapter.SelectCommand := myCommand;
  myDataAdapter.SelectCommand.Connection := myConnection;
  myConnection.Open;
  myDataSet := DataSet.Create;
  myDataAdapter.Fill(myDataSet);
  DataGrid1.DataSource := myDataSet.Tables[0];
  myConnection.close;
end;
```

在上述代码中，首先利用 SqlConnection 对象创建了一个到数据库 MyDatabase 的连接 myConnection，然后通过 SqlCommand 对象执行 SQL 语句“select * from stu”，并把返回结果由 DataAdapter 对象的 Fill 方法填充到 DataSet 对象，最后把结果显示在 DataGrid1 中。

8.4.2 使用 ADO.NET 组件添加数据

在窗体设计中单击“添加数据”标签，将出现“添加数据”标签页的设计界面。在此页中加入若干个 TextBox 组件和 Label 组件，用于添加数据的输入，设计结果见图 8.13。

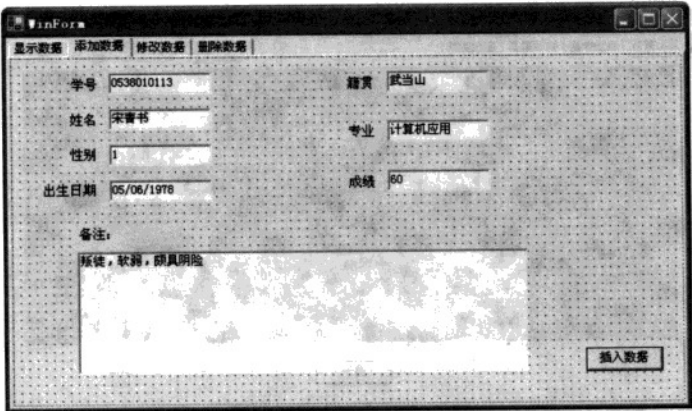


图 8.13 “添加数据”标签页

各组件的属性设置情况如表 8.3 所示。

表 8.3 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TextBox	TextBox1	Text	0538010113
	TextBox2	Text	宋青书
	TextBox3	Text	1
	TextBox4	Text	05/06/1978
	TextBox5	Text	武当山
	TextBox6	Text	计算机应用
	TextBox7	Text	60
	TextBox8	Text	叛徒, 软弱, 颇具阴谋
		Multiline	true
	Label1	Text	学号
	Label2	Text	姓名
	Label3	Text	性别
	Label4	Text	出生日期
	Label5	Text	籍贯
	Label6	Text	专业
	Label7	Text	成绩
Button	Label8	Text	备注:
	Button1	Text	插入数据

属性及界面设计完成以后，编写“插入数据”按钮对应的事件处理过程，代码如下：

```
procedure TWinForm.Button1_Click1(sender: System.Object; e: System.EventArgs);
var SqlStr:string;
```

```
begin
    myConnection := SqlConnection.Create('Data Source = localhost; Integrated Security = SSPI; Initial
Catalog = MyDatabase');
    myCommand := myConnection.CreateCommand;
    SqlStr:= 'Insert into stu values('+TextBox1.Text +', "'+ TextBox2.Text +'", '+ TextBox3.Text +', "'+
TextBox4.Text +', "'+ TextBox5.Text +', "'+ TextBox6.Text +', '+ TextBox7.Text +', "'+ TextBox7.Text +')';
    myCommand.CommandText := SqlStr;
    myConnection.Open;
    myCommand.ExecuteNonQuery;
    myConnection.Close;
end;
```

上面代码中用到的 `myConnection` 和 `myCommand` 对象要在 `private` 部分定义为全局变量，而不能在 `TWinForm.Button1_Click1` 过程中定义为局部变量，否则不能连续进行数据添加操作。定义格式如下：

```
myConnection:SqlConnection;
myCommand:SqlCommand;
```

至此，数据添加模块的界面设计和代码设计任务完成。

8.4.3 使用 ADO.NET 组件更新数据

数据更新也可以用 ADO.NET 中的 `Command` 对象来完成，例如，更改已知姓名的学生成绩可用下列代码实现：

```
myConnection := SqlConnection.Create('Data Source = localhost;
Integrated Security = SSPI; Initial Catalog = MyDatabase');
myCommand := myConnection.CreateCommand;
SqlStr:= 'update stu set grade = '+ TextBox10.Text + ' where name = "'+ TextBox9.Text +'"';
myCommand.CommandText := SqlStr;
myConnection.Open;
myCommand.ExecuteNonQuery;
myConnection.Close;
```

其中，`TextBox9.Text` 用于输入学生的姓名，`TextBox10.Text` 用于输入该学生的成绩。

但是在现实中，人们更多是希望像在 Excel 中那样修改数据，也就是说，人们更习惯于在 `DataGrid` 对象显示的表格中修改数据，然后保存。那么，这里遇到的一个问题就是如何保存在 `DataGrid` 对象中修改过的数据？实际上，这也可以利用 ADO.NET 组件中 `DataAdapter` 对象的 `Update` 方法来完成。

`Update` 方法可以将 `DataSet` 对象中被修改的数据解析回数据源。但是在解析回数据源前，自上次填充 `DataSet` 以来，其他的客户端可能已经修改了数据源中的数据。所以要使用当前数据刷新，再次使用 `DataAdapter` 填充 (`Fill`) `DataSet`。`DataAdapter` 对象的 `Fill` 方法通过检查 `DataSet` 中行的主键值及 `SelectCommand` 返回的行来确定是否要添加一个新行或更新现有行。如果 `Fill` 方法发现 `DataSet` 中某行的主键值与 `SelectCommand` 返回结果中某行的主键值相匹配，则它将用 `SelectCommand` 返回的行中的信息更新现有的行，并将现有行的 `RowState` 属性值设置为 `Unchanged`。如果 `SelectCommand` 返回的行所具有的主键值与 `DataSet` 中行的任何主键值都不匹配，则 `Fill` 方法将添加 `RowState` 为 `Unchanged` 的新行。

为完成在 `DataGrid` 对象中修改数据，在窗体设计器中单击“修改数据”标签项，将出现

该标签页的设计界面, 然后在该页面上放入 DataGrid 组件一个, Button 组件两个, 其 Text 属性值分别设置为“调出数据”(name 属性值为 button2)和“保存修改”(name 属性值为 button5), 其他属性值设为默认值, 结果如图 8.14 所示。

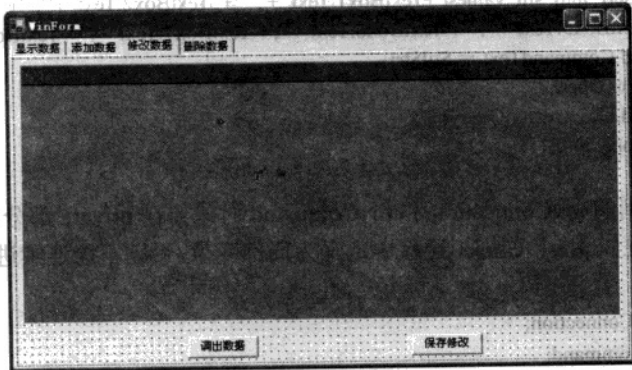


图 8.14 “修改数据”标签页

作为例子, 该例提供修改姓名 (name) 和专业 (speciality) 两列数据的修改功能。在界面设计完成以后, 编写“调出数据”按钮 (Button2) 对应的处理过程, 所得的代码如下:

// “调出数据”(name 属性值为 Button2) 和“保存修改”(name 属性值为 Button5)

procedure TForm.Button2_Click(sender: System.Object; e: System.EventArgs);

begin

//创建连接

myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security info=False;initial catalog=MyDatabase');

//实例化 myDataAdapter 对象, 并建立 SQL 查询语句

myDataAdapter := SqlDataAdapter.Create('select stu_id, name, speciality from stu', myConnection);

//建立 SQL 更新语句

myDataAdapter.UpdateCommand := SqlCommand.Create('update stu set name = @name, speciality = @speciality where stu_id = @stu_id', myConnection);

//设置参数

myDataAdapter.UpdateCommand.Parameters.Add('@name', SqlDbType.NVarChar, 6, 'name');

myDataAdapter.UpdateCommand.Parameters.Add('@speciality', SqlDbType.NVarChar, 50, 'speciality');

mySqlParameter := myDataAdapter.UpdateCommand.Parameters.Add('@stu_id', SqlDbType.Int);

//指定 stu_id 为搜索字段

mySqlParameter.SourceColumn := 'stu_id';

mySqlParameter.SourceVersion := DataRowVersion.Original;

myConnection.Open;

//实例化 myDataSet 对象

myDataSet := DataSet.Create;

//填充 myDataSet 对象

myDataAdapter.Fill(myDataSet, 'stuTable');

//在 DataGrid2 对象中以表格方式显示数据

DataGrid2.DataSource := myDataSet.Tables.Item['stuTable'];

myConnection.close;

end;

然后进一步编写“保存修改”按钮 (Button5) 对应的处理过程, 所得的代码如下:

```

procedure TForm1.Button5_Click(sender: System.Object; e: System.EventArgs);
begin
    // 用 Update 方法可以将 DataSet 对象中被修改的数据解析回数据源
    myDataAdapter.Update(myDataSet.Tables.Item['stuTable']);
end;

```

需要注意的是,采用上述方法修改数据时,至少有一列不能修改,一般是表的主键。在本例中就是 stu_id 字段。

8.4.4 使用 ADO.NET 组件删除数据

删除数据也可以用 ADO.NET 中的 RCommand 对象来完成。比较常用的删除方法是,删除满足一定条件的记录,如删除已知姓名的学生可用下列代码实现:

```

myConnection := SqlConnection.Create('Data Source = localhost;
Integrated Security = SSPI; Initial Catalog = MyDatabase');
myCommand := myConnection.CreateCommand;
SqlStr := 'delete from stu where name = "' + TextBox9.Text + '"';
myCommand.CommandText := SqlStr;
myConnection.Open;
myCommand.ExecuteNonQuery;
myConnection.Close;

```

其中, TextBox9.Text 用于输入学生的姓名。

同样,与对数据更新操作类似,现在人们往往习惯于对 Excel 的操作方式。例如,希望用鼠标来选择某一行,然后删除。为此,在窗体设计器中单击“删除数据”标签项,将出现“删除数据”标签页的设计界面,然后在该页面上放入 DataGrid 组件一个, Button 组件两个,各属性值设置情况如表 8.4 所示。

表 8.4 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Button	Button6	Text	调出数据
	Button8	Text	删除当前行
DataGrid	DataGrid1	ReadOnly	True

适当地调整组件的位置和大小,尽量使布局朴素而美观,如图 8.15 所示。

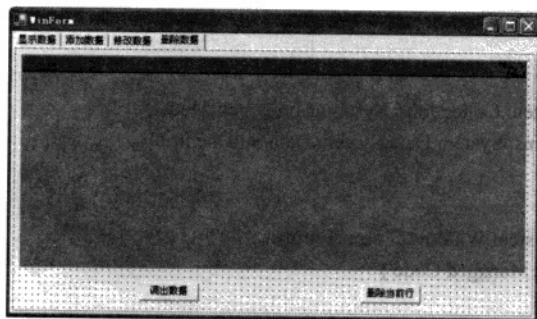


图 8.15 “删除数据”标签页

然后编写两个按钮对应的过程，代码如下：

```

procedure TWinForm.Button6_Click(sender: System.Object; e: System.EventArgs);
begin
    myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security
info=False;initial catalog=MyDatabase');
    myDataAdapter := SqlDataAdapter.Create('select * from stu',myConnection);
    myConnection.Open;
    myDataSet := DataSet.Create;
    myDataAdapter.Fill(myDataSet,'stuTable');
    DataGrid3.DataSource := myDataSet.Tables.Item['stuTable'];
end;

procedure TWinForm.Button8_Click(sender: System.Object; e: System.EventArgs);
var SqlStr,cellValue:string;
    rowIndex:integer;

begin
    rowIndex := DataGrid3.CurrentRowIndex;           //获取当前行的索引号
    cellValue := DataGrid3.Item[rowIndex,0].ToString; //获取当前行的 stu_id 列值
    myCommand := myConnection.CreateCommand;
    SqlStr:= 'delete from stu where stu_id = "' + cellValue + '"'; //建立删除当前行的 SQL 语句
    myCommand.CommandText := SqlStr;
    myCommand.ExecuteNonQuery;                       //执行删除
    myDataSet := DataSet.Create;                      //清除 myDataSet 中的数据，否则会导致显示的数据不断增多
    myDataAdapter.Fill(myDataSet,'stuTable'); //重新填充 myDataSet 对象
    DataGrid3.DataSource := myDataSet.Tables.Item['stuTable']; //在 DataGrid3 对象中显示数据
end;

```

8.4.5 完整的代码设计及程序运行

1. 代码设计

为完整起见，这里给出所有的代码，同时也方便读者学习和比较。这些代码都在单元文件 WinForm.pas 中。代码如下：

```

unit WinForm;

interface

uses
    System.Drawing, System.Collections, System.ComponentModel,
    System.Windows.Forms, System.Data, System.Data.SqlClient;

type
    TWinForm = class(System.Windows.Forms.Form)
    {$REGION 'Designer Managed Code'}
    strict private
        /// <summary>

```

```
/// Required designer variable.  
/// </summary>  
Components: System.ComponentModel.Container;  
TabControl1: System.Windows.Forms.TabControl;  
TabPage1: System.Windows.Forms.TabPage;  
TabPage2: System.Windows.Forms.TabPage;  
TabPage3: System.Windows.Forms.TabPage;  
TabPage4: System.Windows.Forms.TabPage;  
Button0: System.Windows.Forms.Button;  
DataGrid1: System.Windows.Forms.DataGrid;  
Button1: System.Windows.Forms.Button;  
TextBox1: System.Windows.Forms.TextBox;  
TextBox2: System.Windows.Forms.TextBox;  
TextBox3: System.Windows.Forms.TextBox;  
TextBox4: System.Windows.Forms.TextBox;  
TextBox5: System.Windows.Forms.TextBox;  
TextBox6: System.Windows.Forms.TextBox;  
Label1: System.Windows.Forms.Label;  
Label2: System.Windows.Forms.Label;  
Label3: System.Windows.Forms.Label;  
Label4: System.Windows.Forms.Label;  
Label5: System.Windows.Forms.Label;  
Label6: System.Windows.Forms.Label;  
TextBox8: System.Windows.Forms.TextBox;  
Label7: System.Windows.Forms.Label;  
DataGrid2: System.Windows.Forms.DataGrid;  
Button2: System.Windows.Forms.Button;  
Label8: System.Windows.Forms.Label;  
TextBox7: System.Windows.Forms.TextBox;  
Button5: System.Windows.Forms.Button;  
DataGrid3: System.Windows.Forms.DataGrid;  
Button6: System.Windows.Forms.Button;  
Button8: System.Windows.Forms.Button;  
/// <summary>  
/// Required method for Designer support - do not modify  
/// the contents of this method with the code editor.  
/// </summary>  
procedure InitializeComponent;  
procedure Button1_Click(sender: System.Object; e: System.EventArgs);  
procedure Button1_Click1(sender: System.Object; e: System.EventArgs);  
procedure Button2_Click(sender: System.Object; e: System.EventArgs);  
procedure Button5_Click(sender: System.Object; e: System.EventArgs);  
procedure TabPage3_Click(sender: System.Object; e: System.EventArgs);  
procedure Button6_Click(sender: System.Object; e: System.EventArgs);  
procedure Button8_Click(sender: System.Object; e: System.EventArgs);
```

```

{$ENDREGION}
strict protected
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  procedure Dispose(Disposing: Boolean); override;
private
  { Private Declarations }
  var myConnection:SqlConnection;
      myCommand:SqlCommand;
      myDataAdapter:SqlDataAdapter;
      myDataSet:DataSet;
      myDataRow:DataRow;
      mySqlParameter: SqlParameter;
public
  constructor Create;
end;

[assembly: RuntimeRequiredAttribute(TypeOf(TWinForm))]

implementation

{$AUTOBOX ON}

{$REGION 'Windows Form Designer generated code'}
/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWinForm.InitializeComponent;
begin
  Self.TabControl1 := System.Windows.Forms.TabControl.Create;
  Self.TabPage1 := System.Windows.Forms.TabPage.Create;
  Self.DataGrid1 := System.Windows.Forms.DataGrid.Create;
  Self.Button0 := System.Windows.Forms.Button.Create;
  Self.TabPage2 := System.Windows.Forms.TabPage.Create;
  Self.TextBox7 := System.Windows.Forms.TextBox.Create;
  Self.Label8 := System.Windows.Forms.Label.Create;
  Self.Label7 := System.Windows.Forms.Label.Create;
  Self.TextBox8 := System.Windows.Forms.TextBox.Create;
  Self.Label6 := System.Windows.Forms.Label.Create;
  Self.Label5 := System.Windows.Forms.Label.Create;
  Self.Label4 := System.Windows.Forms.Label.Create;
  Self.Label3 := System.Windows.Forms.Label.Create;
  Self.Label2 := System.Windows.Forms.Label.Create;
  Self.Label1 := System.Windows.Forms.Label.Create;

```

```
Self.TextBox6 := System.Windows.Forms.TextBox.Create;
Self.TextBox5 := System.Windows.Forms.TextBox.Create;
Self.TextBox4 := System.Windows.Forms.TextBox.Create;
Self.TextBox3 := System.Windows.Forms.TextBox.Create;
Self.TextBox2 := System.Windows.Forms.TextBox.Create;
Self.TextBox1 := System.Windows.Forms.TextBox.Create;
Self.Button1 := System.Windows.Forms.Button.Create;
Self.TabPage3 := System.Windows.Forms.TabPage.Create;
Self.Button5 := System.Windows.Forms.Button.Create;
Self.Button2 := System.Windows.Forms.Button.Create;
Self.DataGrid2 := System.Windows.Forms.DataGrid.Create;
Self.TabPage4 := System.Windows.Forms.TabPage.Create;
Self.Button8 := System.Windows.Forms.Button.Create;
Self.Button6 := System.Windows.Forms.Button.Create;
Self.DataGrid3 := System.Windows.Forms.DataGrid.Create;
Self.TabControl1.SuspendLayout;
Self.TabPage1.SuspendLayout;
(System.ComponentModel.ISupportInitialize)(Self.DataGrid1).BeginInit;
Self.TabPage2.SuspendLayout;
Self.TabPage3.SuspendLayout;
(System.ComponentModel.ISupportInitialize)(Self.DataGrid2).BeginInit;
Self.TabPage4.SuspendLayout;
(System.ComponentModel.ISupportInitialize)(Self.DataGrid3).BeginInit;
Self.SuspendLayout;
//
// TabControl1
//
Self.TabControl1.Controls.Add(Self.TabPage1);
Self.TabControl1.Controls.Add(Self.TabPage2);
Self.TabControl1.Controls.Add(Self.TabPage3);
Self.TabControl1.Controls.Add(Self.TabPage4);
Self.TabControl1.Dock := System.Windows.Forms.DockStyle.Fill;
Self.TabControl1.Location := System.Drawing.Point.Create(0, 0);
Self.TabControl1.Name := 'TabControl1';
Self.TabControl1.SelectedIndex := 0;
Self.TabControl1.Size := System.Drawing.Size.Create(664, 358);
Self.TabControl1.TabIndex := 0;
//
// TabPage1
//
Self.TabPage1.Controls.Add(Self.DataGrid1);
Self.TabPage1.Controls.Add(Self.Button0);
Self.TabPage1.Location := System.Drawing.Point.Create(4, 21);
Self.TabPage1.Name := 'TabPage1';
Self.TabPage1.Size := System.Drawing.Size.Create(656, 333);
Self.TabPage1.TabIndex := 0;
```

```
Self.TabPage1.Text := '显示数据';
//
// DataGrid1
//
Self.DataGrid1.DataMember := '';
Self.DataGrid1.HeaderForeColor := System.Drawing.SystemColors.ControlText;
Self.DataGrid1.Location := System.Drawing.Point.Create(8, 8);
Self.DataGrid1.Name := 'DataGrid1';
Self.DataGrid1.ReadOnly := True;
Self.DataGrid1.Size := System.Drawing.Size.Create(640, 272);
Self.DataGrid1.TabIndex := 1;
//
// Button0
//
Self.Button0.Location := System.Drawing.Point.Create(288, 296);
Self.Button0.Name := 'Button0';
Self.Button0.TabIndex := 0;
Self.Button0.Text := '显示';
Include(Self.Button0.Click, Self.Button1_Click);
//
// TabPage2
//
Self.TabPage2.Controls.Add(Self.TextBox7);
Self.TabPage2.Controls.Add(Self.Label8);
Self.TabPage2.Controls.Add(Self.Label7);
Self.TabPage2.Controls.Add(Self.TextBox8);
Self.TabPage2.Controls.Add(Self.Label6);
Self.TabPage2.Controls.Add(Self.Label5);
Self.TabPage2.Controls.Add(Self.Label4);
Self.TabPage2.Controls.Add(Self.Label3);
Self.TabPage2.Controls.Add(Self.Label2);
Self.TabPage2.Controls.Add(Self.Label1);
Self.TabPage2.Controls.Add(Self.TextBox6);
Self.TabPage2.Controls.Add(Self.TextBox5);
Self.TabPage2.Controls.Add(Self.TextBox4);
Self.TabPage2.Controls.Add(Self.TextBox3);
Self.TabPage2.Controls.Add(Self.TextBox2);
Self.TabPage2.Controls.Add(Self.TextBox1);
Self.TabPage2.Controls.Add(Self.Button1);
Self.TabPage2.Location := System.Drawing.Point.Create(4, 21);
Self.TabPage2.Name := 'TabPage2';
Self.TabPage2.Size := System.Drawing.Size.Create(656, 333);
Self.TabPage2.TabIndex := 1;
Self.TabPage2.Text := '添加数据';
//
// TextBox7
```

```
//
Self.TextBox7.Location := System.Drawing.Point.Create(368, 112);
Self.TextBox7.Name := 'TextBox7';
Self.TextBox7.TabIndex := 16;
Self.TextBox7.Text := '60';
//
// Label8
//
Self.Label7.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label7.Location := System.Drawing.Point.Create(328, 117);
Self.Label7.Name := 'Label8';
Self.Label7.Size := System.Drawing.Size.Create(40, 23);
Self.Label7.TabIndex := 15;
Self.Label7.Text := '成绩';
//
// Label7
//
Self.Label7.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label7.Location := System.Drawing.Point.Create(64, 166);
Self.Label7.Name := 'Label7';
Self.Label7.TabIndex := 14;
Self.Label7.Text := '备注: ';
//
// TextBox8
//
Self.TextBox7.Location := System.Drawing.Point.Create(64, 190);
Self.TextBox7.Multiline := True;
Self.TextBox7.Name := 'TextBox8';
Self.TextBox7.Size := System.Drawing.Size.Create(440, 120);
Self.TextBox7.TabIndex := 13;
Self.TextBox7.Text := '叛徒, 软弱, 颇具阴险';
//
// Label6
//
Self.Label6.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label6.Location := System.Drawing.Point.Create(328, 67);
Self.Label6.Name := 'Label6';
Self.Label6.Size := System.Drawing.Size.Create(40, 23);
Self.Label6.TabIndex := 12;
Self.Label6.Text := '专业';
//
// Label5
//
```

```
Self.Label5.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label5.Location := System.Drawing.Point.Create(320, 20);
Self.Label5.Name := 'Label5';
Self.Label5.Size := System.Drawing.Size.Create(48, 23);
Self.Label5.TabIndex := 11;
Self.Label5.Text := '籍贯';
//
// Label4
//
Self.Label4.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label4.Location := System.Drawing.Point.Create(29, 123);
Self.Label4.Name := 'Label4';
Self.Label4.Size := System.Drawing.Size.Create(64, 23);
Self.Label4.TabIndex := 10;
Self.Label4.Text := '出生日期';
//
// Label3
//
Self.Label3.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label3.Location := System.Drawing.Point.Create(56, 88);
Self.Label3.Name := 'Label3';
Self.Label3.Size := System.Drawing.Size.Create(40, 23);
Self.Label3.TabIndex := 9;
Self.Label3.Text := '性别';
//
// Label2
//
Self.Label2.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label2.Location := System.Drawing.Point.Create(56, 54);
Self.Label2.Name := 'Label2';
Self.Label2.Size := System.Drawing.Size.Create(32, 23);
Self.Label2.TabIndex := 8;
Self.Label2.Text := '姓名';
//
// Label1
//
Self.Label1.Font := System.Drawing.Font.Create('宋体', 10, System.Drawing.FontStyle.Regular,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label1.Location := System.Drawing.Point.Create(57, 20);
Self.Label1.Name := 'Label1';
Self.Label1.Size := System.Drawing.Size.Create(32, 23);
Self.Label1.TabIndex := 7;
```

```
Self.Label1.Text := '学号';  
//  
// TextBox6  
//  
Self.TextBox6.Location := System.Drawing.Point.Create(368, 63);  
Self.TextBox6.Name := 'TextBox6';  
Self.TextBox6.TabIndex := 6;  
Self.TextBox6.Text := '计算机应用';  
//  
// TextBox5  
//  
Self.TextBox5.Location := System.Drawing.Point.Create(368, 15);  
Self.TextBox5.Name := 'TextBox5';  
Self.TextBox5.TabIndex := 5;  
Self.TextBox5.Text := '武当山';  
//  
// TextBox4  
//  
Self.TextBox4.Location := System.Drawing.Point.Create(96, 120);  
Self.TextBox4.Name := 'TextBox4';  
Self.TextBox4.TabIndex := 4;  
Self.TextBox4.Text := '05/06/1978';  
//  
// TextBox3  
//  
Self.TextBox3.Location := System.Drawing.Point.Create(96, 85);  
Self.TextBox3.Name := 'TextBox3';  
Self.TextBox3.TabIndex := 3;  
Self.TextBox3.Text := '1';  
//  
// TextBox2  
//  
Self.TextBox2.Location := System.Drawing.Point.Create(96, 50);  
Self.TextBox2.Name := 'TextBox2';  
Self.TextBox2.TabIndex := 2;  
Self.TextBox2.Text := '宋青书';  
//  
// TextBox1  
//  
Self.TextBox1.Location := System.Drawing.Point.Create(96, 15);  
Self.TextBox1.Name := 'TextBox1';  
Self.TextBox1.TabIndex := 1;  
Self.TextBox1.Text := '0538010113';  
//  
// Button1  
//
```



```
Self.Button1.Location := System.Drawing.Point.Create(560, 288);
Self.Button1.Name := 'Button1';
Self.Button1.TabIndex := 0;
Self.Button1.Text := '插入数据';
Include(Self.Button1.Click, Self.Button1_Click1);
//
// TabPage3
//
Self.TabPage3.Controls.Add(Self.Button5);
Self.TabPage3.Controls.Add(Self.Button2);
Self.TabPage3.Controls.Add(Self.DataGrid2);
Self.TabPage3.Location := System.Drawing.Point.Create(4, 21);
Self.TabPage3.Name := 'TabPage3';
Self.TabPage3.Size := System.Drawing.Size.Create(656, 333);
Self.TabPage3.TabIndex := 2;
Self.TabPage3.Text := '修改数据';
Include(Self.TabPage3.Click, Self.TabPage3_Click);
//
// Button5
//
Self.Button5.Location := System.Drawing.Point.Create(424, 304);
Self.Button5.Name := 'Button5';
Self.Button5.TabIndex := 4;
Self.Button5.Text := '保存修改';
Include(Self.Button5.Click, Self.Button5_Click);
//
// Button2
//
Self.Button2.Location := System.Drawing.Point.Create(184, 304);
Self.Button2.Name := 'Button2';
Self.Button2.TabIndex := 1;
Self.Button2.Text := '调出数据';
Include(Self.Button2.Click, Self.Button2_Click);
//
// DataGrid2
//
Self.DataGrid2.DataMember := '';
Self.DataGrid2.HeaderForeColor := System.Drawing.SystemColors.ControlText;
Self.DataGrid2.Location := System.Drawing.Point.Create(8, 8);
Self.DataGrid2.Name := 'DataGrid2';
Self.DataGrid2.Size := System.Drawing.Size.Create(640, 280);
Self.DataGrid2.TabIndex := 0;
//
// TabPage4
//
Self.TabPage4.Controls.Add(Self.Button8);
```

```
Self.TabPage4.Controls.Add(Self.Button6);
Self.TabPage4.Controls.Add(Self.DataGrid3);
Self.TabPage4.Location := System.Drawing.Point.Create(4, 21);
Self.TabPage4.Name := 'TabPage4';
Self.TabPage4.Size := System.Drawing.Size.Create(656, 333);
Self.TabPage4.TabIndex := 3;
Self.TabPage4.Text := '删除数据';
//
// Button8
//
Self.Button7.Location := System.Drawing.Point.Create(440, 296);
Self.Button7.Name := 'Button8';
Self.Button7.TabIndex := 4;
Self.Button7.Text := '删除当前行';
Include(Self.Button7.Click, Self.Button8_Click);
//
// Button6
//
Self.Button6.Location := System.Drawing.Point.Create(160, 296);
Self.Button6.Name := 'Button6';
Self.Button6.TabIndex := 1;
Self.Button6.Text := '调出数据';
Include(Self.Button6.Click, Self.Button6_Click);
//
// DataGrid3
//
Self.DataGrid3.DataMember := '';
Self.DataGrid3.HeaderForeColor := System.Drawing.SystemColors.ControlText;
Self.DataGrid3.Location := System.Drawing.Point.Create(8, 8);
Self.DataGrid3.Name := 'DataGrid3';
Self.DataGrid3.ReadOnly := True;
Self.DataGrid3.Size := System.Drawing.Size.Create(640, 272);
Self.DataGrid3.TabIndex := 0;
//
// TWinForm
//
Self.AutoScaleBaseSize := System.Drawing.Size.Create(6, 14);
Self.ClientSize := System.Drawing.Size.Create(664, 358);
Self.Controls.Add(Self.TabControl1);
Self.Name := 'TWinForm';
Self.Text := 'WinForm';
Self.TabControl1.ResumeLayout(False);
Self.TabPage1.ResumeLayout(False);
(System.ComponentModel.ISupportInitialize.SupportInitialize(Self.DataGrid1)).EndInit;
Self.TabPage2.ResumeLayout(False);
Self.TabPage3.ResumeLayout(False);
```

```

    (System.ComponentModel.ISupportInitialize(Self.DataGrid2)).EndInit;
    Self.TabPage4.ResumeLayout(False);
    (System.ComponentModel.ISupportInitialize(Self.DataGrid3)).EndInit;
    Self.ResumeLayout(False);
end;
{$ENDREGION}

```

```

procedure TWinForm.Dispose(Disposing: Boolean);
begin
    if Disposing then
    begin
        if Components <> nil then
            Components.Dispose();
        end;
        inherited Dispose(Disposing);
    end;
end;

```

```

constructor TWinForm.Create;
begin
    inherited Create;
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent;
    //
    // TODO: Add any constructor code after InitializeComponent call
    //
end;

```

```

procedure TWinForm.Button8_Click(sender: System.Object; e: System.EventArgs);
var SqlStr, cellValue: string;
    rowIndex: integer;
begin
    rowIndex := DataGrid3.CurrentRowIndex;
    cellValue := DataGrid3.Item[rowIndex, 0].ToString;
    myCommand := myConnection.CreateCommand;
    SqlStr := 'delete from stu where stu_id = "' + cellValue + '"';
    myCommand.CommandText := SqlStr;
    myCommand.ExecuteNonQuery;
    myDataSet := DataSet.Create;
    myDataAdapter.Fill(myDataSet, 'stuTable');
    DataGrid3.DataSource := myDataSet.Tables.Item['stuTable'];
end;

```

```

procedure TWinForm.Button6_Click(sender: System.Object; e: System.EventArgs);
begin
    myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security
info=False;initial catalog=MyDatabase');

```

```

myDataAdapter := SqlDataAdapter.Create('select * from stu',myConnection);
myConnection.Open;
myDataSet := DataSet.Create;
myDataAdapter.Fill(myDataSet,'stuTable');
DataGrid3.DataSource := myDataSet.Tables.Item['stuTable'];
end;

procedure TWinForm.Button5_Click(sender: System.Object; e: System.EventArgs);
begin
    myDataAdapter.Update(myDataSet.Tables.Item['stuTable']);
end;

procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);
begin
    myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security
info=False;initial catalog=MyDatabase');
    myDataAdapter := SqlDataAdapter.Create('select stu_id, name, speciality from stu',myConnection);
    myDataAdapter.UpdateCommand := SqlCommand.Create('update stu set name = @name,speciality =
@speciality where stu_id = @stu_id', myConnection);
    myDataAdapter.UpdateCommand.Parameters.Add('@name', SqlDbType.NVarChar, 6, 'name');
    myDataAdapter.UpdateCommand.Parameters.Add('@speciality', SqlDbType.NVarChar, 50, 'speciality');
    mySqlParameter := myDataAdapter.UpdateCommand.Parameters.Add('@stu_id', SqlDbType.Int);
    mySqlParameter.SourceColumn := 'stu_id';
    mySqlParameter.SourceVersion := DataRowVersion.Original;
    myConnection.Open;
    myDataSet := DataSet.Create;
    myDataAdapter.Fill(myDataSet,'stuTable');
    DataGrid2.DataSource := myDataSet.Tables.Item['stuTable'];
    myConnection.close;
end;

procedure TWinForm.Button1_Click1(sender: System.Object; e: System.EventArgs);
var SqlStr:string;
begin
    myConnection := SqlConnection.Create('Data Source = localhost; Integrated Security = SSPI; Initial
Catalog = MyDatabase');
    myCommand := myConnection.CreateCommand;
    SqlStr:= 'Insert into stu values('+TextBox1.Text +', "'+ TextBox2.Text +'", '+ TextBox3.Text +', "'+
TextBox4.Text +'", "'+ TextBox5.Text +'", "'+ TextBox6.Text +'",'+ TextBox7.Text +','+ TextBox7.Text +'"');
    myCommand.CommandText := SqlStr;
    myConnection.Open;
    myCommand.ExecuteNonQuery;
    myConnection.Close;
end;

procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
begin
    myConnection := SqlConnection.Create('integrated security=SSPI;data source=localhost;persist security

```

```

info=False;initial catalog=MyDatabase');
myCommand := SqlCommand.Create('select * from stu');
myDataAdapter := SqlDataAdapter.Create;
myDataAdapter.SelectCommand := myCommand;
myDataAdapter.SelectCommand.Connection := myConnection;
myConnection.Open;
myDataSet := DataSet.Create;
myDataAdapter.Fill(myDataSet,'stuTable');
DataGrid1.DataSource := myDataSet.Tables.Item['stuTable'];
myConnection.close;
end;
end.

```

2. 运行程序

代码编写完毕后，运行该程序，默认打开的是“显示数据”标签页。当在该运行界面中单击“显示”按钮时将显示出表 stu 中的所有数据，如图 8.16 所示。

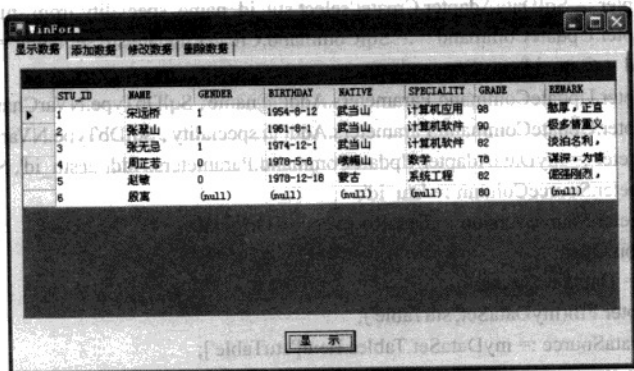


图 8.16 在“显示数据”标签页中显示数据

在界面中单击“添加数据”标签，则会出现“添加数据”标签页。当在相应的文本框中输入数据后，单击“插入数据”按钮即可完成一次数据添加操作，如图 8.17 所示。图中给出的是默认数据，读者可以自由更改。

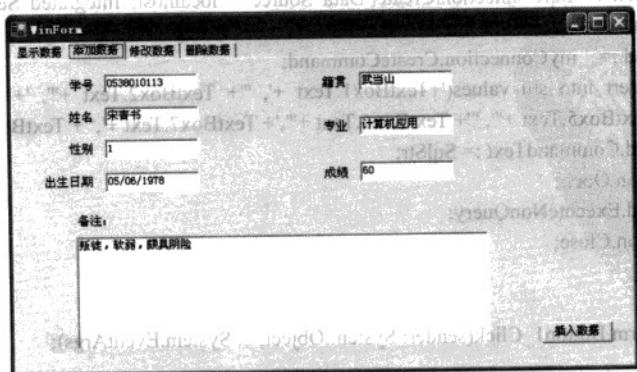


图 8.17 在“添加数据”标签页中插入数据

当单击“修改数据”标签时,则会出现“修改数据”标签页,如图8.18所示。在此页中,可以在 DataGrid 对象显示的表格中直接修改数据,非常符合人们的操作习惯。但作为例子,本例只提供 name 和 speciality 列值的修改功能,读者完全可以模仿该方法来完成所有列值的修改。

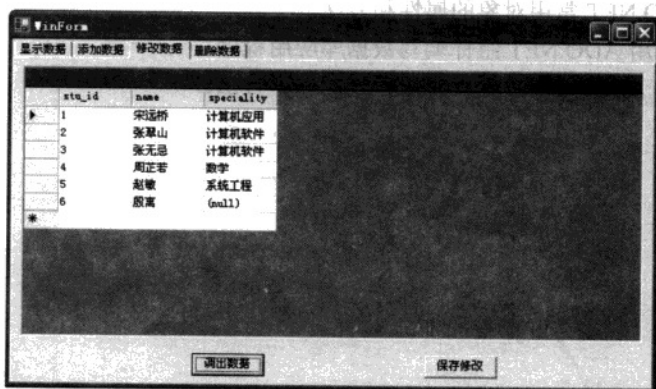


图 8.18 在“修改数据”标签页中更新数据

程序 ADOInsUpdDel 的最后一项功能是数据的删除。数据的删除和数据的修改在编程实现中非常类似,在此不作重复介绍。但考虑到人们的一些操作习惯——“用鼠标选择某一行,然后整行删除”,在删除模块中主要是实现这一功能。

单击“删除数据”标签,将出现“删除数据”标签页。首先单击“调出数据”按钮,调出表 stu 中的所有数据,如果要删除某一行,如第四行,则选择第四行,如图 8.19 所示,然后单击“删除当前行”按钮,第四行数据就会从数据库中删除。删除操作可以不断重复进行,直到没有数据为止。

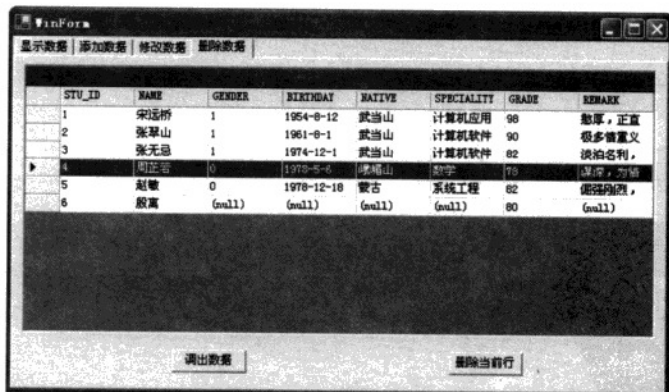


图 8.19 在“删除数据”标签页中删除数据

8.5 小结

众所周知,微软是软件界的霸主,其推出的.NET 技术已经覆盖各个领域,具有广阔的应

用前景。作为.NET 的核心技术之一,ADO.NET 技术在本章中已经进行了较为全面的介绍,并辅以具体的实例开发。希望学完本章后,读者能掌握下列要点:

- ADO.NET 技术的特点。
- 熟悉 ADO.NET 常用对象的属性和方法。
- 熟练地运用 ADO.NET 组件编写数据库应用程序。
- 掌握开发 ADO.NET 数据库应用程序的基本方法。

第9章 基于 dbExpress 的数据库应用开发

dbExpress 和 BDE 是 Delphi 推出的两个重要的数据库引擎。不同的是,在发布 BDE 数据库应用程序时,要同时发布 BDE 数据库引擎,这使 BDE 显得笨重;而 dbExpress 可实现跨平台的数据访问功能,具有快速、简便、容易发布等特点。本章主要是对 dbExpress 技术及其应用进行较为全面的介绍,内容要点涉及:

- dbExpress 的特点。
- 常用 dbExpress 组件的属性和方法,及其数据集的单向性。
- dbExpress 数据库的连接方法。
- dbExpress 数据库的数据浏览方法、查询设计和数据管理的技术,包括数据插入、删除和更新。
- dbExpress 数据库应用程序开发的基本步骤。

9.1 了解 dbExpress

dbExpress 是 Borland 公司开发的一种数据访问技术,它提供了与数据库平台无关的数据存取功能,可以实现数据的动态 SQL 处理。dbExpress 是从 Delphi 6 以后推出的数据访问技术,其开发的初衷是打算将其作为 Linux 中 Kylix 的数据访问层,但现在它已成为用于 Delphi、C++Builder 和 Kylix (既包括 Delphi 版本也包括 C++版本)的功能强大的跨平台数据访问层。

dbExpress 为各种数据库提供了驱动程序,这些驱动程序以独立动态库的形式存在(在 Windows 下是.dll 文件,在 Linux 下是.so 文件)。Borland 还提供了 MySQL 和 Interbase 驱动程序源代码,这使数据库厂商根据需要扩充和开发高性能的驱动程序。

与 BDE、ADO.NET 相比,dbExpress 是一种轻量级的数据库引擎。它只需要两个共享的对象库,即 dbExpress 驱动程序和 LIBMIDAS.SO,这可有效地最小化应用程序的规模、简化安装,使得 dbExpress 成为一种快速、简便、容易发布的对象。其具体特点和优势体现在:

- dbExpress 可以进行跨平台访问,如在 Windows 操作系统中可以 CLX 组件实现跨平台访问功能。
- 可以有效地实现真正的分布式数据库开发,这可以通过与 DataSnap 相结合的方法来实现。
- dbExpress 可以有效地节约系统资源,因为它在数据访问过程中没有在内存开辟缓冲区,使用的是单向、只读的数据集;另外,与 BDE 相对而言,dbExpress 发布时所占的空间要小得多。
- dbExpress 在运行时较其他数据库引擎更加有效,这是因为当用户定义的查询在数据库服务器上执行时,dbExpress 使用远程数据查询或者其他外部查询,而不产生内部查询。
- 具有通用 API 的优点,而没有 BDE 等数据库引擎的额外开销。

总之, dbExpress 被设计成为一种快速、轻便并且容易开发的数据库引擎, 但一般适用于较小的数据处理, 它通常用于编写数据库服务器的客户端软件。

dbExpress 的实现仅仅需要五个接口 (Interface): ISQLDriver、SQLConnection、SQLCommand、SQLCursor 和 SQLMetaData, 它们是都在 dbExpress.h 中定义, 可以直接使用 Object Pascal 编写实现程序。但是 ISQLDriver 不能用 C++ 代码来实现, 必须使用 SQLDriver 接口。

这几个核心接口的功能介绍如下:

- ISQLDriver (SQLDriver)。这种接口用于进行特殊的数据库初始化操作, 如初始化环境、分配句柄等, 并且可以用于获取一个 SQLConnection 接口。
- SQLConnection。该接口用于建立一个到数据库的连接, 还可以获得用来查询和处理存储过程的 SQLCommand 对象, 或者获得用来追溯源数据的 SQLMetaData 对象和事务处理。
- SQLCommand。该接口提供了用于处理查询或存储数据的方法, 支持多次执行一个被准备好的参数已绑定的查询, 并能从一个存储过程返回多个游标。
- SQLCursor。一个查询或存储过程执行完毕后将返回有关数据或源数据, SQLCursor 接口则用于获取这些返回结果; 此外, 它还提供一些用于获取个体域的信息及其属性值的方法。
- SQLMetaData。SQLMetaData 是一种源数据访问接口, 它用于追溯不同的数据库的源数据。但 SQLMetaData 提供的源数据仅适用于 Borland 公司开发数据访问组件, 如 Delphi 和 C++ Builder 中的数据库访问组件。

为了实现数据库访问的跨平台特性, dbExpress 定义了自己的逻辑数据类型, 如表 9.1 所示。

表 9.1 窗体及各组件属性的设置

逻辑数据类型	含义
fldUNKNOWN	不确定的字段类型
fldZSTRING	字符串类型
fldBOOL	布尔类型
fldNT16	16 位带符号数
fldNT32	32 位带符号数
fldFLOAT	浮点数
fldBLOB	Blob 类型
fldTIME	时间类型
fldDATETIME	日期类型
fldFMTBCD	BCD 类型

目前, dbExpress 支持的数据库并不是很多, 主要限于六种数据库: Oracle 9i、SQL Server 2000、My SQL 3.23.49、DB2、InterBase 6.5/6.x 和 Inoformix SE。

9.2 建立第一个 dbExpress 数据库应用程序

下面通过实例来说明创建 dbExpress 数据库应用程序的基本方法。实例创建的步骤如下：

(1) 首先创建一个 VCL Form 应用程序，方法是：在 Delphi 2005 IDE 中选择菜单 File→New→VCL Form Application 菜单命令，命名为 FirstdbExpressApp。

(2) 在窗体中添加组件 SQLConnection、SQLQuery、DataSource 和 DBNavigator 各一个及八个 DBEdit 组件。

(3) 设置各组件的相关属性，结果如表 9.2 所示。

表 9.2 窗体及各组件属性的设置

组件类别	组件名称	属性设置项目	设置结果
SQLConnection	SQLConnection1	ConnectionName	MSSQLConnection
		DriverName	MSSQL
		GetDriverFunc	getSQLDriverMSSQL
		Params	单击其右边的  按钮，打开 Value List editor 对话框，并进行设置，如图 9.1 所示
		LoginPrompt	False (使得登录数据库时不弹出登录对话框)
		Connected	True
SQLQuery	SQLQuery1	SQLConnection	SQLConnection1
		SQL	select * from employee
		Active	True
DataSource	DataSource1	DataSet	SQLQuery1
DBNavigator	DBNavigator1	DataSource	DataSource1
DBEdit	DBEdit1	DataSource	DataSource1
		DataField	fname
	DBEdit2	DataSource	DataSource1
		DataField	hire_date
	DBEdit3	DataSource	DataSource1
		DataField	pub_id
	DBEdit4	DataSource	DataSource1
		DataField	minit
	DBEdit5	DataSource	DataSource1
		DataField	job_lvl
	DBEdit6	DataSource	DataSource1
		DataField	lname
	DBEdit7	DataSource	DataSource1
		DataField	job_id

续表

组件类别	组件名称	属性设置项目	设置结果
	DBEdit8	DataSource	DataSource1
		DataField	emp_id
Form	Form1	Caption	第一个 dbExpress 数据库应用程序

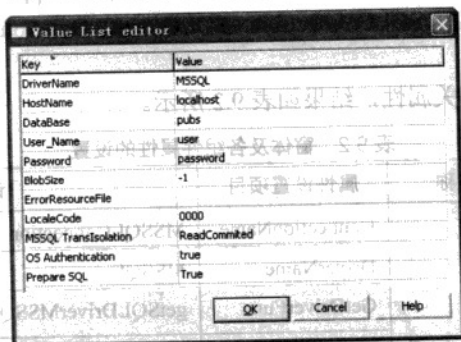


图 9.1 Value List editor 对话框 (设置 Params 属性)

(4) 适当地调整窗体中各组件的大小和位置, 结果如图 9.2 所示。

(5) 属性值和界面设计完后, 运行 FirstdbExpressApp 程序, 如图 9.3 所示。



图 9.2 FirstdbExpressApp 程序窗体的设计界面

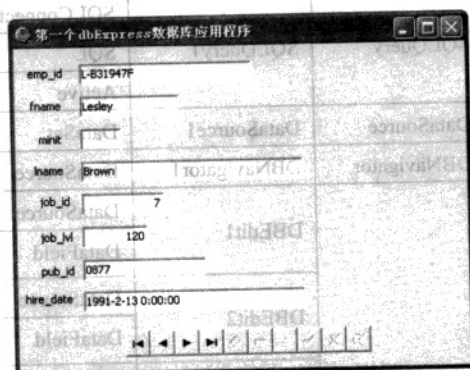


图 9.3 FirstdbExpressApp 程序的运行界面

在运行界面中, 可以通过单击 按钮使得记录指针移到第一条记录上, 或者通过单击 按钮使得记录指针移到下一条记录上。但除了这两个按钮外, 单击其他按钮都会产生错误。原因在于, 所有的数据集型的 dbExpress 组件都是单向的, 而不能进行双向浏览。下一节将专门对常用的 dbExpress 组件进行介绍, 通过进一步学习还会发现对 dbExpress 组件不能直接进行编辑, 需要另寻它法才能完成数据的更新、删除和插入操作。

显然, 与 BDE 类似, dbExpress 也是通过其组件来实现对数据库的访问。因此, 要深入探讨 Delphi 应用程序的开发, 必须对 dbExpress 组件有较为深入的了解。

9.3 熟悉常用的 dbExpress 组件

由上节可知,dbExpress 数据应用程序的开发离不开 dbExpress 组件。本节专门就 dbExpress 组件的属性和方法进行介绍。

从工具面板中的 dbExpress 标签中可以看到,dbExpress 组件一共包含七个,如图 9.4 所示。它们可以分为两类:数据库连接组件和数据集组件,以下分别讲述主要的几个组件。

9.3.1 TSQLConnection 组件

TSQLConnection 组件是数据集组件和数据库之间连接的桥梁,它与 TDatabase 组件在 BDE 中的功能相似,dbExpress 利用它可以实现数据库的封装与后端数据库的连接,从而完成对数据的存取任务。

对于 TSQLConnection 组件,它有两个重要的配置文件:dbxconnections.ini 和 dbxdrivers.ini,它们都是文本文件,位于 C:\Program Files\Common Files\Borland Shared\DBExpress 目录下。其中,dbxconnections.ini 包含 dbExpress 的所有命名连接及这些连接的具体信息,如驱动器类型、数据库文件、URL 等。下面是 dbxconnections.ini 文件中连接 MySQLConnection 的具体定义信息:

```
[MySQLConnection]
DriverName=MySQL
HostName=ServerName
Database=DBNAME
User_Name=user
Password=password
BlobSize=-1
ErrorResourceFile=
LocaleCode=0000
```

在程序运行时,也可以动态指定自己定义的 dbxconnections.ini 文件,方法是:把 TSQLConnection 组件的 LoadParamsOnConnect 属性设置为 True,然后把注册表的“KEY_CURRENT_USER\Software\Borland\DBExpress”目录下的 Connection Registry File 项的值改为指向自己定义的 dbxconnections.ini 文件的位置。这样,程序运行时将根据 Connection Registry File 项的值来读取自己定义的 dbxconnections.ini 文件,从而达到更换 dbxconnections.ini 文件的目的。

dbxdrivers.ini 文件则包含 dbExpress 所支持的驱动器及驱动器具体设置的列表,如驱动程序文件、用户名和密码等。下面的代码是 dbxdrivers.ini 文件中的部分内容:

```
[DB2]
GetDriverFunc=getSQLDriverDB2
LibraryName=dbexpdb2.dll
VendorLib=db2cli.dll
Database=DBNAME
User_Name=user
```



图 9.4 dbExpress 组件

```
Password=password  
BlobSize=-1  
ErrorResourceFile=  
LocaleCode=0000  
DB2 TransIsolation=ReadCommitted
```

```
[Interbase]  
GetDriverFunc=getSQLDriverINTERBASE  
LibraryName=dbexpint.dll  
VendorLib=gds32.dll  
Database=database.gdb  
RoleName=RoleName  
User_Name=sysdba  
Password=masterkey  
ServerCharSet=  
SQLDialect=1  
BlobSize=-1  
CommitRetain=False  
WaitOnLocks=True  
ErrorResourceFile=  
LocaleCode=0000  
Interbase TransIsolation=ReadCommitted
```

```
[MySQL]  
GetDriverFunc=getSQLDriverMYSQL  
LibraryName=dbexpmysql.dll  
VendorLib=libmysql.dll  
HostName=ServerName  
Database=DBNAME  
User_Name=user  
Password=password  
BlobSize=-1  
ErrorResourceFile=  
LocaleCode=0000
```

```
[Oracle]  
GetDriverFunc=getSQLDriverORACLE  
LibraryName=dbexpora.dll  
VendorLib=oci.dll  
Database=Database Name  
User_Name=user  
Password=password  
RowsetSize=20  
BlobSize=-1  
ErrorResourceFile=  
LocaleCode=0000
```

Oracle TransIsolation=ReadCommitted

[Informix]

GetDriverFunc=getSQLDriverINFORMIX

LibraryName=dbexpinf.dll

VendorLib=isqlb09a.dll

HostName=ServerName

Database=Database Name

User_Name=user

Password=password

BlobSize=-1

ErrorResourceFile=


LocaleCode=0000

Informix TransIsolation=ReadCommitted

从上面的这个 dbxdrivers.ini 文件中可以看出, 对于 DB2 类型的数据库, 其进入点函数为 getSQLDriverDB2, 驱动库文件为 dbexpdb2.dll, 提供厂商文件为 db2cli.dll 等。

与 dbxconnections.ini 文件类似, 可以通过修改注册表的“KEY_CURRENT_USER\Software\Borland\DBExpress”目录下的 Driver Registry File 项值的方法来达到动态更改 dbxconnections.ini 文件的目的。

每一个数据集组件都有一个 SqlConnection 属性, 如果多个数据集组件的 SqlConnection 属性值均指向同一个 TSQLConnection 组件, 那么可以实现数据库的共享。TSQLConnection 组件常用的属性并不多, 主要包括以下几种:

- (1) ConnectionName 属性。用于指定选用的数据库引擎。
- (2) Connected 属性。布尔类型, 如果组件已经连接, 则返回 true, 否则返回 false。
- (3) ConnectionState 属性。返回当前组件的连接状态。
- (4) LoginPrompt 属性。布尔类型, 如果为 true 则在登录数据库时显示登录对话框, 否则不显示。
- (5) Params 属性。用于设置连接参数, 设置方法是: 单击该属性项右边的省略号按钮 , 打开 Value List editor 对话框, 如图 9.5 所示。

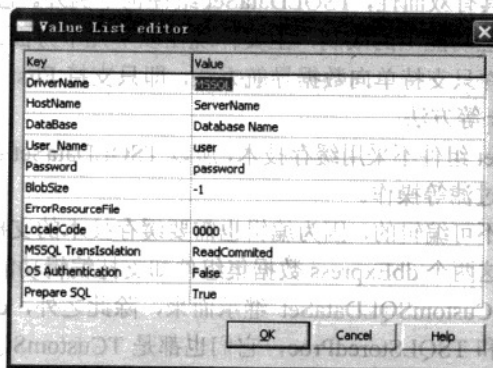


图 9.5 Value List editor 对话框

在此对话框中主要设置以下几项:

- **DriverName**: 设置驱动程序名称。
- **HostName**: 设置 SQL 数据库服务器名称。
- **DataBase**: 设置数据库实例名称。
- **User_Name**: 设置登录名。
- **Password**: 设置登录密码。
- **OS Authentication**: 当该属性值为 true 时, 表示使用系统验证登录方式, 即将以 Windows 的用户名和密码作为登录用户名和密码 (推荐), 否则要根据提供的 User_Name 和 Password 值来验证。

(6) **LibraryName** 属性。建立连接使用的 dbExpress 驱动程序, 一般在 **ConnectionName** 和 **DriverName** 属性设置后, 这项会自动设置。

(7) **OnLogin** 事件。当 **LoginPrompt** 为 true 时, 如果登录数据库将会触发该事件, 否则 (**LoginPrompt** 为 false) 不会触发该事件。

(8) **Execute** 方法。该方法的原型为:

```
Function Execute(const SQL:string; Params:TParams):integer
```

它用于执行带参数的 SQL 语句, 但是并不是返回数据集而是返回受影响的记录的数目。

对于不带参数的 SQL 语句, 可用 **ExecuteDirect** 方法来执行, 其原型为:

```
Function ExecuteDirect (const SQL:string):LongWord;
```

如果执行成功, 该函数将返回 0, 否则返回 dbExpress 的错误代码。

9.3.2 TSQLDataSet 组件

TSQLDataSet 是 dbExpress 的数据集组件, 该数据集组件是单向的, 即返回的数据集的记录指针是单向的, 只能向前移动而不能向后。所以, dbExpress 的数据集不涉及双向游标、记录缓冲等昂贵开销的技术。可见, dbExpress 最大的优势就是轻量级的, 与之相比, BDE 就显得很“笨重”。BDE 提供完整的编辑功能、数据集的向前向后导航及其多个后端数据库的驱动功能等。但是, 具备这些功能要付出的代价是: 一方面, 安装 BDE 大约需要 10M 的内存空间; 另一方面, BDE 本身也会占用大量的内存资源, 而且在传输时数据流量也很大。

实际上, 任何特性都具有双面性, **TSQLDataSet** 组件也不例外。它的优势是给数据访问带来方便的同时也导致了一些功能上的限制, 主要体现在以下几个方面:

- **TSQLDataSet** 组件只支持单向数据导航方法, 即只支持 **First** 方法和 **Next** 方法, 而不支持 **Last** 和 **Prior** 等方法。
- 由于 **TSQLDataSet** 组件不采用缓存技术, 所以 **TSQLDataSet** 组件无法进行字段查找, 也不能进行记录过滤等操作。
- **TSQLDataSet** 是不可编辑的, 因为编辑也需要缓存技术的支持。但 **TClientDataSet** 和 **TSimpleDataSet** 这两个 dbExpress 数据集组件却支持编辑功能。

TSQLDataSet 是从 **TCustomSQLDataSet** 继承而来, 除此之外, dbExpress 数据集还包括 **TSQLQuery**、**TSQLTable** 和 **TSQLStoredProc**, 它们也都是 **TCustomSQLDataSet** 的派生类, 如图 9.6 所示。

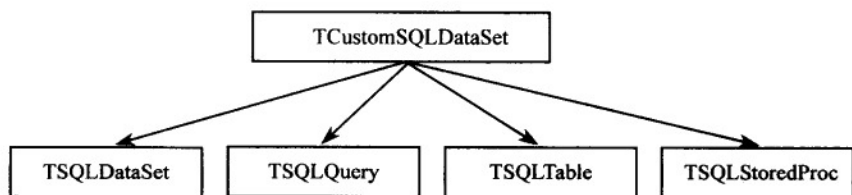


图 9.6 dbExpress 数据集及其继承关系

而 TCustomSQLDataSet 则是类 TDataSet 的派生类，此外，TDataSet 的派生类还包括 TCustomClientDataSet 和 TBDEDataSetSet。

TSQLDataSet 是典型的 dbExpress 数据集，它完全可以代替其他的 dbExpress 数据集。实际上，其他的 dbExpress 数据集也都具有与 TSQLDataSet 相似的属性、方法和事件等。例如，它们都只能使用 Next 方法和 First 方法，而不能使用 Prior 方法及 Last 方法。TSQLDataSet 属性和方法都是从 TCustomSQLDataSet 继承而来的，说明如下：

(1) CommandText 属性。其值是字符串型，但是具体作用和意义由 CommandType 属性值决定。

(2) CommandType 属性。该属性的可能取值有三种，即 ctQuery、ctStoredProc、ctTable。如果取值为 ctQuery，则表示 CommandText 属性值要设置为 SQL 语句；如果取值为 ctStoredProc，则要调用存储过程；如果取值为 ctTable，则表示要获取一个数据表中的数据，CommandText 属性值应为表名。

(3) SQLConnection 属性。用于设置连接的 SQLConnection。

(4) DataSource 属性。设置本数据集的数据源。

(5) Params 属性。获取执行查询或者存储过程的参数（如果需要的话）。

(6) RecordCount 属性。返回数据集中记录的总数。

(7) ExecSQL 方法。执行没有返回结果集的 SQL 语句或存储过程。

(8) GetFieldData 方法。GetFieldData 方法用于获取字段值，与之有相同作用的是字段的 GetData 方法。

(9) GetKeyFieldNames 方法。获取数据集中所有索引使用的字段名称列表。

(10) ParamByName 方法。通过参数名获取命令中的查询参数值。

9.3.3 TSQLQuery 组件

TSQLQuery 组件用于执行 SQL 语句，它的应用比较灵活，可以表示一个结果集的对象（如 Select 返回的结果集），也可以用于执行那些没有返回结果集的 SQL 语句（如 Delete、Insert 和 Update 等）。该组件可以在设计阶段时静态创建，也可以在程序运行时动态创建。TSQLQuery 也是一种单向的数据集，只能使用 Next 方法和 First 方法移动记录指针，它不支持对数据直接进行编辑，而是要寻找其他方法来实现。TSQLQuery 组件的主要属性和方法如下：

(1) Active 属性。布尔类型，当其值为 true 时表示激活当前的数据集。

(2) CommandText 和 CommandType 属性。这两个属性分别与 TSQLDataSet 组件对应的属性一样，CommandText 属性值的具体作用和意义由 CommandType 属性值决定。当

CommandType 属性取值为 ctQuery, 则表示 CommandText 属性值要设置为 SQL 语句; 如果取值为 ctStoredProc, 则要调用存储过程; 如果取值为 ctTable, 则表示要获取一个数据表中的数据, CommandText 属性值应为表名。

(3) DataSource 属性。用于把当前数据集连接到另一个数据集。

(4) MaxBlobSize 属性。MaxBlobSize 属性返回从 Blob 类型字段中所能获取的最大字节数。

(5) Params 属性。Params 属性用于设置在一个查询或存储过程中用到的参数。

(6) SQLConnection 属性。用于指定连接到数据库的 SQLConnection 对象。

(7) Open 方法和 Close 方法。Open 方法用于激活当前数据集, 其作用与把 Active 属性设置为 true 时一样; Close 方法则用于关闭当前数据集, 相当于把 Active 属性设置为 false。

(8) First 方法。该方法用于把记录指针移到第一条记录上。

(9) Next 方法。Next 方法使记录指针向下移一条记录。

(10) Free 方法。Free 方法用于撤销当前的数据集并释放其占用的资源。

另外, 需要注意的是, 对于 TSQLQuery 组件, Edit、Insert、UpdateRecord 等方法只是“形同虚设”, 实际上是不能直接使用的(虽然编译可以通过)。

9.3.4 TSQLTable 组件


TSQLTable 组件与 TTable 的作用类似, 但它的功能较之后者则弱得多。它也是一个单向的数据集, 只能运用 First 和 Next 方法移动记录指针, 而不能采用 Prior 或 Insert 等方法, 不支持对数据直接进行编辑。其主要属性和方法如下:

- SQLConnection 属性。SQLConnection 属性值用于连接到数据库的 SQLConnection 对象。
- Active 属性。当该属性的值设置为 true 时, 表示激活数据集。
- TableName 属性。TableName 属性值用于指定相应数据库中的数据表。
- Eof 属性。布尔类型, 如果当前记录指针指向最后一条记录, 则 Eof 属性返回 true, 否则返回 false。
- IndexName 属性。用于指定一个已经在数据库中定义过的索引, 使得数据集中的记录按照该索引来排序, 同时作为从属表连接到主表的关联字段。
- IndexFieldName 属性。IndexFieldName 属性与 IndexName 属性的功能类似, 但不同的是在 IndexFieldName 属性值中可以输入一个或者多个索引名, 名字之间用分号隔开。
- RecordCount 属性。返回当前数据集中的记录数。
- MasterFieldst 属性。设置连接到主表的字段(如果需要的话)。
- MasterSource 属性。指定主表使用的数据源。

9.4 熟悉 dbExpress 数据库的连接方法

所有的 dbExpress 数据集组件都要通过 TSQLConnection 组件来连接数据库, 而 TSQLConnection 组件则通过 ConnectionName 属性的设置来创建数据库连接。连接数据库的方法可以分为两种: 一种是使用已经存在的数据库连接, 即修改已有的连接, 使之连接到既定的数据库; 另一种是创建新的数据库连接。

9.4.1 使用已有的数据库连接

对于使用已有的数据库连接，其操作过程非常容易。首先在窗体中放入一个 TSQLConnection 组件，然后单击该 ConnectionName 属性项右边的下拉列表按钮，在 Delphi 2005 中会出现如图 9.7 所示的下拉列表框。该列表框列出了所有已有的数据库连接，只要选择相应的选项即可完成对 ConnectionName 属性的设置。

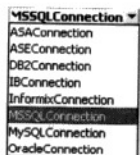



图 9.7 设置 ConnectionName 属性

但需要注意的是，设置了 ConnectionName 属性以后并没有真正建立数据库连接，而是需要进一步对 TSQLConnection 进行修改和配置。配置的方法是：在对象观察器中找到 TSQLConnection 组件的 Params 属性项并单击该项右边的省略号按钮，打开 Value List editor 对话框。要在此对话框中进行必要的设置，如数据库服务器名称、数据库实例名等。可以看出，各项的意义还比较直观，设置较为容易，如连接 SQL Server 数据库中的 pubs 实例的设置情况如图 9.8 所示。

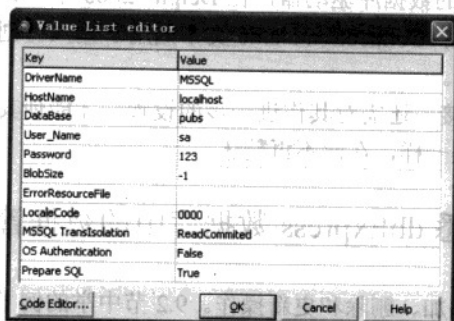


图 9.8 Value List editor 对话框（连接 pubs）

当 ConnectionName 属性设置完毕以后，发现 DriverName、GetDriverfunc、LibraryName 等，会自动被设置。实际上，这些默认值都是由 dbxdrivers.ini 文件指定的。

在一个数据库连接设置好以后，把 Connected 属性值设置为 true，这时将弹出用户名和密码登录框；只要正确输入相应的用户名和密码，如果连接有效则将出现连接成功的提示信息，否则会给出连接错误的原因。如果不希望在每次连接时都出现登录框，只要把 LoginPrompt 属性设置为 false 即可，这时连接将使用 Params 属性中指定的用户名（User Name）和密码（Password）。

9.4.2 创建新的数据库连接

上一节介绍了修改已有数据库连接的方法。但已有的连接往往已被既定的应用程序所使用，对其进行修改往往使得相应的程序无法运行，所以在更多情况下还是希望重新创建一个数据库连接。

创建一个新的数据库连接时，可以按照下列步骤完成：

- (1) 在应用程序的窗体中放入 TSQLConnection 组件，然后右击该组件，在弹出的快捷

菜单中选择 Edit Connection Properties 命令, 将会出现连接编辑对话框, 如图 9.9 所示。

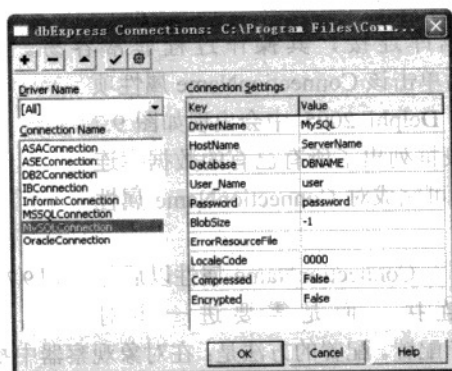


图 9.9 连接编辑对话框

(2) 单击对话框左上方的 **+** 按钮, 将弹出新连接对话框。其中, Driver Name 下拉列表框将显示 dbExpress 所支持的数据库驱动器, 在 Delphi 2005 中一共有八种数据库驱动器, 用户可根据需要从中选择一种; Connection Name 文本则用于输入待创建的数据库连接的名称。设置完毕后, 单击 OK 按钮。

(3) 对于新创建的连接, 还需对其作进一步的设置, 才能使其成为真正有效的连接。设置的方法与上节介绍的方法一样, 在此不再赘述。

9.5 熟悉 dbExpress 数据库中的数据管理技术

数据管理是指数据的添加、删除和更新操作。9.2 节中举的例子实际上就是数据的浏览, 在此不再重复。需要注意的是, 对于 dbExpress 数据库不能用 DBGrid 组件来浏览数据。

前面已经指出, dbExpress 的数据集组件不提供数据添加、更新和删除功能, 因此为实现数据的基本管理, 需要另辟它法。一般有两种方法: 一种是把数据集提供器组件 TDataSetProvider 和客户数据集组件 TClientDataSet 搭配使用, 在客户端创建双向数据集 TClientDataSet, 使之复制单向数据集的一个副本, 形成数据集的缓存, 然后在这个缓存中进行数据的添加、更新和删除等操作; 另一种是使用 dbExpress 组件面板中的 TSimpleDataSet 组件。

9.5.1 使用 dbExpress 数据集组件

每一种 dbExpress 数据集组件都提供了有关的属性和方法, 通过利用这些属性和方法可以实现对数据的管理。

- TSQLDataSet 组件。首先要把它 CommandType 属性设置为 ctQuery, 然后使要执行的 SQL 语句以字符串的形式赋给 TSQLDataSet 组件的 CommandText 属性; 如果 CommandText 属性被设置为 ctStoredProc, 则 CommandText 属性应设置为要执行的存储过程。
- TSQLQuery 组件。把要执行的 SQL 语句以字符串的形式直接赋给它的 SQL 属性。

- TSQLStoredProc 组件。它只能执行已定义的存储过程，方法是在 StoredProc 属性中设置相应的存储过程代码。

当对上述组件的有关属性进行设置后，还需对其发出相应的执行命令，这样才能实现对数据的操作。TSQLDataSet 组件和 TSQLQuery 组件都有方法 ExecSQL，当执行这个方法的时候，设置的 SQL 语句才真正起作用。该方法原型如下：

```
Function ExecSQL(ExecDirect: Boolean = False): Integer; override;
```

该方法有一个布尔型参数 ExecDirect，其默认值为 False。当参数 ExecDirect 设置为 True 时，表示在执行 SQL 语句之前不必事先做好准备；如果其值为 False 则表示要事先准备 SQL 语句。

对于 TSQLStoredProc 组件，则要执行 ExecProc 方法。该方法原型如下：

```
Function ExecProc: Integer; virtual;
```

可见，该方法是一个整型函数，其返回值表示该方法执行后受到影响的记录数，同时该记录数也被反映到属性 RowsAffected 中。该方法既可以用来执行具有返回数据的存储过程，也可以用以执行没有数据返回的存储过程。

前面说过，dbExpress 数据集组件是单向只读组件，它们能够执行的只是那些没有返回结果的 SQL 语句，如 insert、delete、update 等。在执行过程中，dbExpress 数据集组件只是将 SQL 语句传递给相应的数据库服务器，而不做任何检查。

9.5.2 使用 TSQLConnection 组件

TSQLConnection 组件也有一个执行 SQL 语句的方法，即 Execute 方法。该方法可以直接把 SQL 语句发送到数据库服务器。该方法原型如下：

```
Function Execute(const SQL: string; Params: TParams; ResultSet: Pointer = nil): Integer;
```

该方法含有三个参数，其中参数 SQL 是 SQL 语句构成的子串；Params 是 TParams 类对象，用于传递 SQL 语句中包含的参数（如果有参数的话，否则该参数设置为 nil）；参数 ResultSet 可以缺省，它是指向 TCustomSQLDataSet 类型变量的指针。当执行的 SQL 语句有返回数据时，Execute 方法将自动创建一个 TCustomSQLDataSet 类对象，并使 ResultSet 指向该对象；如果没有返回数据时则该参数的值被设置为 nil。

因为 SQL 语句具有对数据库进行添加、删除和修改等功能，所以使用上述两种方法都可以进行有效的数据管理。

9.5.3 使用 ClientDataSet 组件

数据的添加、删除和更新等操作还可以用 ClientDataSet 组件提供的方法来完成，这些方法包括 insert、append、edit、post 等。其基本原理是，在客户端使用 ClientDataSet 从 dbExpress 数据集组件中获取数据并创建一个新的数据集副本，实现数据的缓存功能，通过对缓存区中的数据进行操作，使得一个单向的、只读的数据集变成可读可写的数据集。例如，对于数据更新，可以采用 ClientDataSet 组件的 edit 方法和 post 方法来完成，前者用于把 ClientDataSet 设置成为编辑状态，以使得可以修改其中的数据；修改完成以后，调用 post 方法提交。根据这种思想，分别对两个按钮（假设为 button1 按钮和 button2 按钮）编写事件处理程序：

```
procedure TForm1.Button1Click(Sender: TObject);
```

```

begin
    ClientDataSet1.Edit;
    Button1.Enabled := false;
    Button2.Enabled := true;
end;
procedure TForm1.Button2Click(Sender: TObject);
begin
    ClientDataSet1.Post;
    Button2.Enabled := false;
    Button1.Enabled := true;
end;

```

这样，当单击 button1 按钮后，通过相应的方法修改缓存中的数据，然后单击 button2 按钮，保存已做出的修改。

9.6 dbExpress 数据库开发实例


本节将通过开发一个实例来说明如何实现 dbExpress 数据库中的数据管理，包括数据查询、添加、删除和更新等操作。

首先创建一个名为 dbExpressInsUpdDel.dpr 的工程，其窗体对应的单元文件采用默认的名称 Unit1.pas。

9.6.1 界面设计和属性设置

在工程 dbExpressInsUpdDel 的窗体中放入下列组件：SQLConnection、SQLDataSet、ClientDataSet、DataSource 和 DataSetProvider 各一个，以及 GroupBox 组件三个和 Button 组件七个，并对它们的属性进行设置，结果如表 9.3 所示。

表 9.3 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
SQLConnection	SQLConnection1	ConnectionName	MSSQL.Connection
		DriverName	MSSQL
		GetDriverFunc	getSQLDriverMSSQL
		Params	单击其右边的  按钮，打开 Value List editor 对话框，并进行设置，使之连接到 MyDatabase 数据库实例
		LoginPrompt	False (使得登录数据库时不弹出登录对话框)
		Connected	True
SQLDataSet	SQLDataSet1	SQLConnection	SQLConnection1
		CommandText	ctTable
		CommandType	stu
DataSource	DataSource1	DataSet	ClientDataSet1

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DataSetProvider	DataSetProvider1	DataSet	SQLDataSet1
ClientDataSet	ClientDataSet1	所有属性均使用默认值	
GroupBox	GroupBox1	Caption	数据插入
	GroupBox2	Caption	更新数据
	GroupBox3	Caption	删除数据
Button	Button1	Caption	在当前记录插入
	Button2	Caption	在末尾插入
	Button3	Caption	设置更新状态
	Button4	Caption	保持更新结果
	Button5	Caption	删除当前记录
	Button6	Caption	删除所有数据
	Button7	Caption	保存插入数据
Form	Form1	Caption	dbExpress 数据库应用程序: 数据插入、删除和更新

属性和界面设计完成以后，结果如图 9.10 所示。



图 9.10 运行程序 dbExpressInsUpdDel.dpr 的设计界面

9.6.2 代码设计——实现数据添加、更新和删除功能

1. 数据添加

分别编写“在当前记录插入”(Button1)、“在末尾插入”(Button2)和“保存插入数据”(Button7)按钮的事件处理程序，结果如下：

```
procedure TForm1.Button1Click(Sender: TObject); // Button1
begin
```

```
ClientDataSet1.Insert;
Button1.Enabled := false;
Button7.Enabled := true;
end;

procedure TForm1.Button2Click(Sender: TObject); // Button2
begin
    ClientDataSet1.Append;
    Button7.Enabled := true;
    Button2.Enabled := false;
end;

procedure TForm1.Button7Click(Sender: TObject); // Button7
begin
    ClientDataSet1.Post;
    Button7.Enabled := false;
    Button1.Enabled := true;
    Button2.Enabled := true;
end;
```

2. 数据删除

删除数据分为两种情况：一种是删除当前的记录，另一种是删除所有的数据。对于后者，一般在删除之前首先给出一个删除提示，以免造成灾难性后果。具体的删除操作可以由 ClientDataSet 事件的 delete 方法来完成。基于这样的思考，编写“删除当前记录”（Button5）和“删除所有数据”（Button6）按钮的事件处理程序，结果如下：

```
procedure TForm1.Button5Click(Sender: TObject);
begin
    ClientDataSet1.Delete; //删除当前记录
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
    if MessageDlg('确认要删除所有的数据吗?', mtWarning, [mbYes, mbNo], 0) = mrNo then exit;
    while not ClientDataSet1.Eof do ClientDataSet1.Delete; //删除所有数据
end;
```

3. 数据更新

更新数据可以采用 ClientDataSet 组件的 edit 方法和 post 方法来完成，前者用于把 ClientDataSet 设置成为编辑状态，以使得可以修改其中的数据；修改完成以后，调用 post 方法提交。根据这种思想，分别对“设置更新状态”（Button3）和“保持更新结果”（Button4）按钮对应的事件处理过程编写代码，结果如下：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
    ClientDataSet1.Edit;
    Button3.Enabled := false;
    Button4.Enabled := true;
```



```

end;

procedure TForm1.Button4Click(Sender: TObject);
begin
    ClientDataSet1.Post;
    Button4.Enabled := false;
    Button3.Enabled := true;
end;

```

代码编写完成以后, 编译、运行 dbExpressInsUpdDel.dpr 程序, 结果如图 9.11 所示。



图 9.11 dbExpressInsUpdDel.dpr 程序运行界面

在该运行界面中, 可以完成记录的添加、更新和删除操作。

9.6.3 dbExpress 数据库中的查询设计

回顾前面相关的内容可知, 不管是使用 dbExpress 数据集组件还是使用 TSQLConnection 组件, 它们执行的都是不带返回数据的 SQL 语句。所以, 探讨在 dbExpress 数据库中如何实现查询功能就显得很重要。

一般来说, 大多数的查询设计都是用带 where 子句的 select 语句来完成的。本节中, 对上节创建的 dbExpressInsUpdDel 程序加以补充, 使之具有查询功能。具体地讲, 使之具有按每一个字段进行查询的功能。

为此, 首先在 dbExpressInsUpdDel 程序的窗体中加入两个 GroupBox 组件, 一个 Edit 组件、一个 Button 组件和一个 ComboBox 组件。其中 ComboBox 组件用于选择字段名 (由程序自动把每一个字段的名称加入到 GroupBox 中), Edit 组件用于输入要查询的关键字, Button 组件则用于执行查询操作。补加的这几个组件的属性设置情况如表 9.4 所示。

表 9.4 各补加组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
GroupBox	GroupBox4	Caption	数据查询
Button	Button9	Caption	查询

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Edit	Edit1	Text	请输入查询关键词
ComboBox	ComboBox1	Text	ClientDataSet1
		Items	请选择字段名
		style	csDropDownList

属性设置完毕后，适当调整新添加组件的大小和位置，结果如图 9.12 所示。

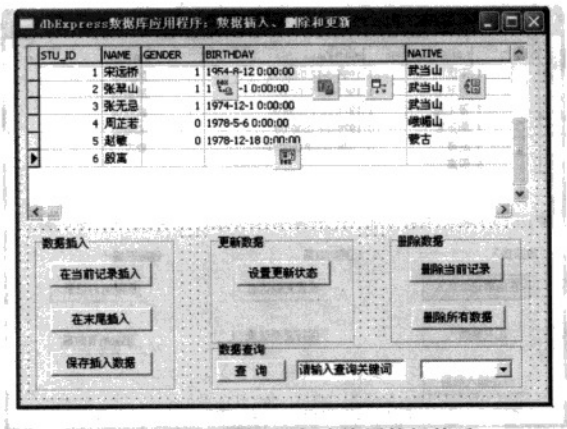


图 9.12 设计界面（添加查询用的组件后）

在编码时，首先让程序自动获取每一个字段的名称并把它们添加到 ComboBox 组件中。完成这一功能的代码显然要放在窗体的 FormCreate 过程中，代码如下：

```
procedure TForm1.FormCreate(Sender: TObject);
var str:string;
    iFieldCount,i:integer;
begin
    ComboBox1.ItemIndex := 0;
    iFieldCount := SQLQuery1.FieldCount;
    for i := 0 to iFieldCount-1 do
    begin
        str:= SQLQuery1.Fields[i].FieldName;
        ComboBox1.Items.Add(str);
    end;
end;
```

然后编写“查询”（Button9）按钮的事件处理代码，代码如下：

```
procedure TForm1.Button9Click(Sender: TObject);
var sqlStr:string;
begin
    if (Edit1.Text = '请输入查询关键词') or (ComboBox1.Text = '请选择字段名') then
    begin
```

```
ShowMessage('请输入查询关键词和选择字段名');
exit;
end;

sqlStr := 'select * from stu where ' + ComboBox1.Text + ' = "' + Edit1.Text + '"';
SqlQuery1.SQL.Text := sqlStr;
SqlQuery1.Close;
ClientDataSet1.Close;
SqlQuery1.ExecSQL(); //执行查询操作
SqlQuery1.Open;
ClientDataSet1.Open;
DBGrid1.Refresh;
end;
```

这样,就使得程序 dbExpressInsUpdDel 多增加了一个查询功能:按每一个字段进行查询。运行该程序,在组合框中选择“NAME”(表示按姓名查询),在编辑框中输入“赵敏”,然后单击“查询”按钮,在 DBGrid 组件中将列出姓名为“赵敏”的所有学生(本例只有一个),如图 9.13 所示。

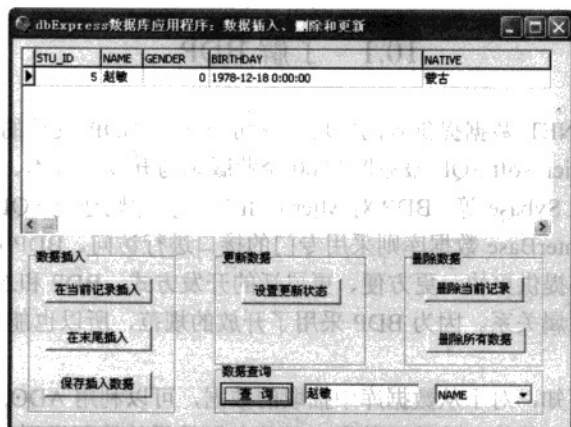


图 9.13 查询姓名为“赵敏”的学生

9.7 小结

dbExpress 是 Delphi 提供的数据库开发重要技术之一,它可以实现跨平台的数据访问功能,而且 dbExpress 具有快速、简便、容易发布等特点,在许多应用中都可以发挥其特有的优点。本章对 dbExpress 技术进行了较为全面的介绍,学完后读者应该达到下列目标:

- 了解 dbExpress 和 BDE 的区别,包括特点和使用区别。
- 熟悉常用的 dbExpress 组件及其数据集的单向性。
- 掌握 dbExpress 数据库的连接方法。
- 掌握 dbExpress 数据库的数据浏览方法、查询设计和数据管理的基本技术,包括数据插入、删除和更新。
- 掌握 dbExpress 数据库应用开发的一般方法。

第 10 章 基于 BDP 的数据库应用开发

BDP (Borland Data Provider) 提供了一种高性能的数据访问架构和统一的编程模型, 用 BDP 开发的程序易于发布, 具有开发速度快、过程简洁等特点。本章主要介绍基于 BDP 的数据库应用开发技术, 涉及的要点包括:

- BDP 数据库的特点。
- 常用 BDP 组件 (如 BdpConnection、BdpCommand、BdpDataAdapter 等) 的基本属性和方法。
- 基于 BDP 的数据浏览技术。
- 基于 BDP 的数据管理, 包括数据添加、数据更新和数据删除等。

10.1 了解 BDP

BDP 组件是一个 .NET 数据提供者的实现, 它可以访问 BDP 支持的任何数据库。Delphi 2005 中的 BDP 支持 Microsoft SQL 服务器 2000 企业版本与开发者版本、Oracle 10g、Borland InterBase、IBM DB2 及 Sybase 等。BDP 对 Microsoft 的数据库则通过 SQL OLE DB 层来访问, 而对 Oracle、DB2 和 InterBase 数据库则采用专门的接口进行访问。BDP 可作为大量数据处理的企业级应用, 它能够提供更快速、更方便、更灵活的开发方式。BDP 和 .NET 框架的数据提供者是并列的, 而不是隶属关系。因为 BDP 采用了开放的规范, 所以也能为其他数据库创建第三方驱动器。

通过前面的学习可知, 对于从数据库中抽取的数据, 可以利用 ADO.NET 的 DataSet 组件来组织和管理, 体现了一种统一的处理数据表及其之间关系的编程模型。与此类似, 在 BDP 中, 也同样可以使用 DataSet 组件来组织和使用 BDP 组件从数据源中获取的数据。

BDP 中继承了 .NET 框架的逻辑数据类型, 提供了一个内置的映射区支持数据库。可以通过扩张 BDP 的方法并通过实现一个 .NET 提供者接口来支持其他的数据库管理系统。 .NET 框架可以赋予对每个数据源创建单独的提供者的能力, Delphi 则进一步提供了一套通用的方法来简化这项支持多种数据库的任务, 那就是基于 BDP 接口的 BDP 数据源插件技术, 通过插件来实现对多种数据源的访问。与完全创建一个新的数据提供者来说, 创建插件就容易得多了。

BDP 的特点可以体现在以下几个方面:

- 提供了一个高性能的数据访问架构和统一的编程模型, 容易扩展并应用于多种数据库平台。
- 采用了开放的规范, 易为其他数据库创建第三方驱动器。
- 集成度高, 非常容易连接到任何 BDP 支持的数据库。
- BDP 不需要 COM Interop 层。
- 具有通用于各种数据库的数据类型。

10.2 熟悉常用的 BDP 组件

BDP 组件与 ADO.NET 组件非常相似，也主要分为连接组件和数据集组件。

10.2.1 BdpConnection 组件

BDP 组件位于工具面板中的 Borland Data Provider 标签页上。下面从 BdpConnection 组件开始介绍常用的 BDP 组件。

BdpConnection 组件的主要属性和方法如下：

(1) **ConnectionString** 属性。BdpConnection 组件用于连接数据库，它包含的属性不多，最主要的就是 **ConnectionString** 属性。该属性设置的方法是：在对象观察器中单击该属性右边的下拉列表按钮 ▾，在弹出的下拉列表框中将列出当前所有的连接，如图 10.1 所示。

一般来说，这些连接还不能使用，需要进一步的配置。方法是：在窗体设计器中右击 BdpConnection 组件，在弹出的快捷菜单中选择 **Connection Editor ...** 命令，将打开数据库连接编辑器，如图 10.2 所示。



图 10.1 数据库的连接

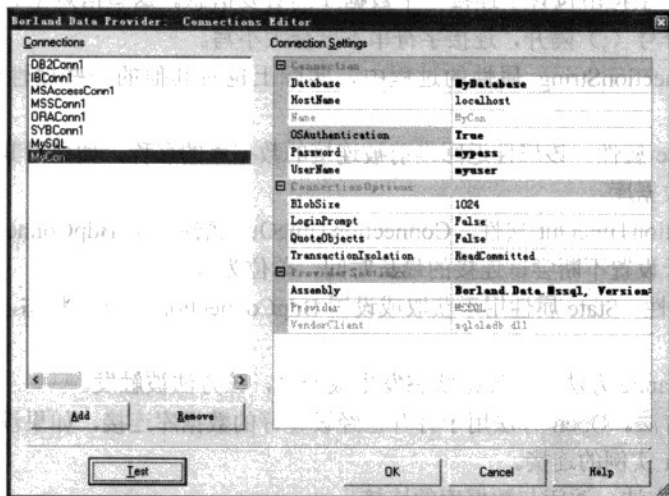


图 10.2 数据库连接编辑器

在数据库连接编辑器中可以对已有的连接进行修改（配置），也可以通过单击 **Add** 按钮重新创建一个连接并配置它。例如，在本例中单击 **Add** 按钮时弹出 **Add New Connection** 对话框。在下拉列表框中根据数据库类型选择相应的提供者名称，如 **MSSQL**；然后单击 **OK** 按钮回到数据库连接编辑器界面，在此进一步设置新创建连接的属性，主要包括以下几项：

- **Database**：该项用于设置要连接数据库的名称。
- **HostName**：该项则指定数据库服务器的名称。
- **OSAuthentication**：该项为布尔类型，当设置为 **true** 时表示采用操作系统的验证方式。

这时 Password 和 UserName 项无效。

- Password: 设置登录数据库的密码, 但有效的前提是 OSAuthentication 为 false。
- UserName: 设置登录数据库的用户名, 但有效的前提是 OSAuthentication 为 false。

设置完毕以后, 可以单击 Test 按钮测试新创建的数据库连接是否成功。在图 10.2 所示的界面中, 创建的连接 MyCon 则是连接到了本地机上 SQL Server 中的 MyDatabase 数据库实例, 该系统采用操作系统验证方式, 所以 UserName 和 Password 无效, 建议使用这种方式。

实际上, 上面那么多步的操作仅相当于给 BdpConnection 组件的 ConnectionString 属性值赋下列的字符串:

```
'assembly=Borland.Data.Mssql,
  Version=2.0.0.0,
  Culture=neutral,
  PublicKeyToken=91d62ebb5b0d1b1b;
vendorclient=sqloledb.dll;
osauthentication=True;
database = MyDatabase;
username=myuser;
hostname = localhost;
password = mypass;
provider = MSSQL'
```

显然, 这一个字符串包含了连接一个数据库的必要信息。这些信息由若干个关键属性组成, 它们之间用分号 (;) 隔开, 连接字符串不区分大小写。

在设置 ConnectionString 属性的过程中, 实际上也对其他的一些属性进行了设置, 如 Database 属性等。

(2) Database 属性。该属性返回当前被连接的数据库的名称, 如果对其进行设置则表示要连接到某一个数据库。

(3) ConnectionTimeout 属性。ConnectionTimeout 属性也是 BdpConnection 组件的一个重要属性, 它用于设置不断尝试连接的最长时间, 单位为秒。

(4) State 属性。State 属性用于获取或设置 BdpConnection 组件的状态, 支持两个值, 即 Open 和 Closed。

(5) StateChange 方法。当连接状态发生改变时, 该方法被触发。

(6) Open 方法。Open 方法用于打开已经设置好的数据库连接, 如果没有则它会创建一个新的 SQL Server 实例的连接。

(7) Closed 方法。关闭与数据库的连接。

10.2.2 BdpCommand 组件

BdpCommand 组件主要用于执行一些 SQL 语句, 包括用于返回数据、修改数据、运行存储过程以及发送或检索参数信息的数据库命令。如果创建了 BdpDataAdapter 对象则会自动创建 BdpCommand 对象。

BdpCommand 对象常用的属性和方法包括:

(1) CommandText 属性。定义命令的可执行文本, 包括 SQL 语句、存储过程等, 要与 CommandType 属性搭配使用。

(2) CommandType 属性。CommandType 属性用于决定 CommandText 属性值的解释方式。当 CommandType 属性值取 Text 时, CommandText 属性值为 SQL 语句, 表示要直接执行这些 SQL 语句; 当 CommandType 属性值取 StoredProcedure 时, 则 CommandText 属性值被解释为存储过程名, 表示要调用存储过程; 当 CommandType 属性值取 TableDirect 时, 则 CommandText 属性值为要读取的数据表名。

CommandType 属性的默认值为 Text。

(3) CommandTimeOut 属性。用于设置或返回终止执行命令之前需要等待的时间 (单位为秒), 默认值为 0。

(4) Connection 属性。用于设置或返回数据库的连接对象, 在执行 BdpCommand 之前必须设置该属性。

(5) DesignTimeVisible 属性。该属性值是布尔型, 当它取值为 false 时, BdpCommand 组件在设计阶段也是不可见的, 默认是 true。

(6) Parameters 属性。有些命令在执行时需要参数, Parameters 属性则用于传递给命令对象的参数, 它是一个 BdpParameterCollection 集合。其默认值是一个空集合。

(7) ParametersCount 属性。ParametersCount 属性返回组件中参数的个数。

(8) Cancel 方法。该方法用于取消命令的执行, 回到执行前的状态。

(9) CreateParameter 方法。该方法用于生成命令的参数。

(10) ExecuteNonQuery 方法。该方法用于执行 SQL 语句, 但这些语句没有返回结果集, 如 Insert、Update、Delete 等。例如下列代码可以实现对数据库 northwind 中表 Customers 的数据插入:

```
myConnection := SqlConnection.Create('Data Source = localhost;  
                                     Integrated Security = SSPI;  
                                     Initial Catalog = northwind');  
myCommand := myConnection.CreateCommand;  
myCommand.CommandText :=  
    'Insert into Customers(CustomerID, CompanyName) values("A11", "Lenovo");'  
myConnection.Open;  
myCommand.ExecuteNonQuery;
```

(11) ExecuteReader 方法。执行有返回结果的 SQL 语句 (Select)、存储过程等, 返回结果存放在 DataReader 对象中。

(12) CreateXMLReader 方法。该方法与 ExecuteReader 方法类似, 不同的是其返回结果将以 XML 数据形式保存到 XMLReader 对象中。

(13) Prepare 方法。该方法用于在数据库实例上创建命令的一个准备 (或编译) 命令版本。

(14) ResetCommandTimeOut 方法。用于恢复 CommandTimeOut 属性的默认值。

10.2.3 BdpDataAdapter 组件

BdpDataAdapter 组件是 DataSet 组件和数据源之间的桥梁, 是 BDP 的数据适配器, 其功能类似于 ADO.NET 的 DataAdapter 组件。BdpDataAdapter 组件要与 DataSet 对象结合起来使用才能完成对数据库的访问。BdpDataAdapter 组件的具体作用体现在两个方面: 一个是用来填充数据集 (DataSet); 另一个是更新数据源。

当要填充数据集的时候, 它将调用 Fill 方法, 但在此之前必须设置它的 SelectCommand 属性, 以用于指示从数据库中获取的数据的结构; 当要更新数据源的时候, 则要事先有效地设置 BdpDataAdapter 组件的 InsertCommand、UpdateCommand 或 DeleteCommand 属性。可见, 利用 BdpDataAdapter 组件可以方便地实现数据的查询和更新操作, 因此 BdpDataAdapter 组件在访问数据库过程中扮演着十分重要的角色。

BdpDataAdapter 组件常用的属性和方法主要包括:

(1) Active 属性。用于激活 BdpDataAdapter 组件, 当 Active 属性值被设置为 true 时, 被激活。BdpDataAdapter 组件被激活时, 执行 SelectCommand 返回的结果将被填充到数据集组件 (DataSet) 中。

(2) DataSet 属性。返回或设置 BdpDataAdapter 组件对应的数据集对象。

(3) SelectCommand 属性。SelectCommand 属性是一个 Command 对象 (下面讲到的 InsertCommand 属性、UpdateCommand 属性和 DeleteCommand 属性等也都是 Command 对象), 用于从数据源中抽取数据。实际上该属性存储的是 SQL 的 Select 命令, 用于从数据库中抽取数据, 并存于 DataSet 中。数据存入 DataSet 的过程实际上就是调用 Fill 方法的过程, 所以在调用 Fill 方法之前要先设置该属性。

同样也可以对之赋予一个已创建的 BdpCommand 对象, BdpDataAdapter 组件将按照 BdpCommand 对象的具体设置来执行数据的抽出任务。注意, 当把一个已创建的 BdpCommand 对象赋给 SelectCommand 属性时, BdpCommand 对象不会被复制, SelectCommand 属性只是维护一个 BdpCommand 对象的引用。这对于 InsertCommand、UpdateCommand 或 DeleteCommand 属性也有类似的情况。

(4) InsertCommand 属性。InsertCommand 属性用于把数据从 DataSet 对象中插入到数据源 (数据库) 中。该属性存储的是 SQL 的 Insert 命令。可以多次将该属性赋予不同的值, 使之能够完成多种数据插入操作。

与此相类似, 也可以把一个已创建的 BdpCommand 对象赋给 InsertCommand 属性, BdpDataAdapter 组件将按照 BdpCommand 对象的具体设置来执行数据的插入任务。

(5) UpdateCommand 属性。用于更新和修改数据源中的数据, 数据是来自于 DataSet。该属性存储的是 SQL 的 Update 命令。当执行 UpdateCommand 属性中设置的命令时, 如果返回了数据库中受影响的记录, 这些记录可能被填充到 DataSet 中, 但这要取决于 BdpCommand 对象的 UpdatedRowSource 属性值。

(6) DeleteCommand 属性。用于删除数据源中满足既定条件的数据。该属性存储的是 SQL 的 Delete 命令。可以多次给该属性赋予不同的值, 从而完成不同的数据删除任务。

与 InsertCommand 属性和 UpdateCommand 属性类似, 可以对之赋予一个已创建的 BdpCommand 对象, BdpDataAdapter 组件将按照 BdpCommand 对象的具体设置来执行数据的删除任务。

(7) MaxRecords 属性。返回或设置填充一个 DataSet 对象时能够使用的最大记录数, 默认值为 100。

(8) StartRecord 属性。设置基底记录的序号, 即记录开始的序号, BdpDataAdapter 填充 DataSet 时从这个序号开始, 默认值为 0。

(9) FillError 方法。当 BdpDataAdapter 填充 DataSet 过程中出现错误时, 该方法被触发。

(10) RowUpdating 方法。当 BdpDataAdapter 更新开始, 但还没有完成时该方法被触发。

(11) RowUpdated 方法。当 BdpDataAdapter 更新并完成后该方法被触发。

(12) Fill()方法。Fill()方法用于把 SelectCommand 返回的数据填充到 DataSet。填充过程分为两步: 首先通过 BdpDataAdapter 的 SelectCommand 属性从数据库中抽出需要的数据, 然后再利用 BdpDataAdapter 的 Fill()方法把抽出来的数据填充到 DataSet。上文已经指出, DataSet 实际上是多个 DataTable 对象的集合, 所以填充 DataSet 实际上是意味着填充 DataSet 中的 DataTable 对象。因此, 在调用 Fill()方法时要以 DataSet 对象名为参数。

10.2.4 BdpDataReader 组件

BdpDataReader 组件也可以从数据源中抽取数据, 但与 BdpDataAdapter 组件不同的是, 它是只读数据流, 而不能对其进行写操作。BdpDataReader 组件也必须与 BdpConnection 组件搭配使用才能实现相应的数据抽取功能, 下面介绍其主要属性和方法。

(1) FieldCount 属性。FieldCount 属性用于获取或者设置当前行的字段数。

(2) IsClosed 属性。该属性值是布尔类型, 返回数据读取器的状态。当数据读取器关闭时其值为 true 时, 否则为 false。

(3) Items 属性。Items 属性用于设置或者返回字段值。通过该属性, 可以非常方便地访问 BdpDataReader 组件所包含的数据。具体访问方法可以通过字段名称, 也可以通过字段序号来实现。

(4) RecordsAffected 属性。RecordsAffected 属性值为整型数值, 表示对 BdpDataReader 进行数据更新、插入、删除等操作时受到影响的记录数量。当操作失败或者没有记录受到操作的影响时, 该属性值返回 0; 而在执行 Select 命令时, 该属性值则返回-1。

10.3 熟悉基于 BDP 的数据浏览技术

在 Windows Form 应用程序中, 浏览数据最常用的组件就是 DataGrid。下面通过一个实例来介绍如何在“后台”使用 BDP 组件而“前台”运用 DataGrid 组件来浏览数据库中数据的方法。

该实例是一个 Windows Form 应用程序, 命名为 BDPViewData.dpr, 它可以显示局域网内任一台 SQL Server 服务器上的任一个数据库中的任一张数据表。

10.3.1 创建 BDP 对象

浏览数据只需要两个 BDP 组件和一个 DataSet 组件, 这两个 BDP 组件分别是 BdpConnection 和 BdpDataAdapter。

首先, 在 Delphi 2005 IDE 中创建一个 Windows Form 应用程序, 命名为 BDPViewData.dpr。然后分别创建 BdpConnection、BdpDataAdapter 和 DataSet 对象, 方法是在工具面板中找到相应的组件并双击它们即可。

另外, 还在窗体中添加下列组件: DataGrid 和 Button 组件各一个, Label 和 TextBox 组件各三个。设置各个组件的属性, 其结果如表 10.1 所示。

表 10.1 各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Button	Button1	Text	显示数据
Label	Label1	Text	数据库服务器名:
	Label2	Text	数据库名:
	Label3	Text	数据表名:
TextBox	TextBox1	Text	localhost
	TextBox2	Text	MyDatabase
	TextBox3	Text	stu
BdpConnection	BdpConnection1	这三个组件只要添加到窗体设计器中就行 (相当于创建相应的对象), 不用作任何设置	
BdpDataAdapter	BdpDataAdapter1		
DataSet	DataSet1		
DataGrid	DataGrid1	在设计阶段其所有属性均使用默认值	

适当地调整它们的位置和大小, 结果如图 10.3 所示。

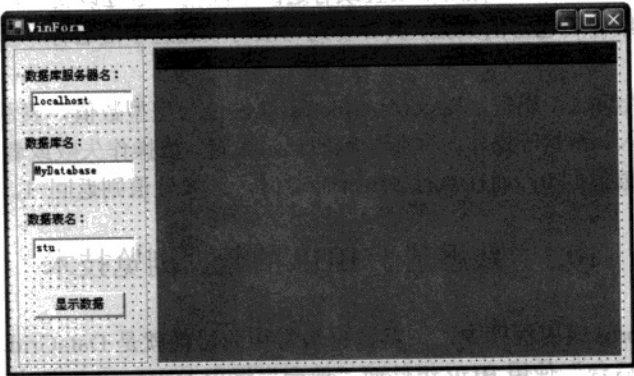


图 10.3 程序 BDPViewData.dpr 的设计界面

10.3.2 开发 BDP 对象

界面设计完成以后, 即可对 BDP 对象进行编码了。

(1) 首先要从三个 TextBox 组件中分别取出数据库服务器名、数据库名和表名, 并分别置于相应的变量中:

```
hostname := TextBox1.Text;  
database := TextBox2.Text;  
tablename := TextBox3.Text;
```

(2) 用代码设置 BdpConnection1 对象的 ConnectionString 属性, 使之连接到其数据库和数据表:

```
BdpConnection1.ConnectionString := 'assembly=Borland.Data.Mssql,  
Version=2.0.0.0,
```

```

Culture=neutral,
PublicKeyToken=91d62ebb5b0d1b1b;
vendorclient=sqloledb.dll;
osauthentication=True;
database='+ database +';
hostname='+ hostname +';
provider=MSSQL';

```

(3) 在 BdpDataAdapter1 对象中构造查询语句:

```
SqlStr := 'select * from '+tablename;
```

```
BdpDataAdapter1.SelectCommand.CommandText := SqlStr;
```

```
BdpDataAdapter1.SelectCommand.Connection := BdpConnection1;
```

(4) 填充 DataSet1 对象, 并在 DataGrid1 中显示数据:

```
DataSet1.Tables.Clear;
```

```
try
```

```
    BdpDataAdapter1.Fill(DataSet1);
```

```
    DataGrid1.DataSource := DataSet1.Tables[0];
```

```
except on e: Exception do
```

```
    MessageBox.Show(e.Message);
```

```
end;
```

显然, 这部分操作是在“显示数据”按钮(Button1)对应的过程中完成的, 所得代码如下:

```
procedure TWinForm.Button1_Click1(sender: System.Object; e: System.EventArgs);
```

```
var hostname,database,tablename,SqlStr:string;
```

```
begin
```

```
    hostname := TextBox1.Text;
```

```
    database := TextBox2.Text;
```

```
    tablename := TextBox3.Text;
```

```
    SqlStr := 'select * from '+tablename;
```

```
BdpConnection1.ConnectionString := 'assembly=Borland.Data.Mssql,
```

```
    Version=2.0.0.0,
```

```
    Culture = neutral,
```

```
    PublicKeyToken = 91d62ebb5b0d1b1b;
```

```
    vendorclient = sqloledb.dll;
```

```
    osauthentication = True;
```

```
    database = '+ database +';
```

```
    hostname = '+ hostname +';
```

```
    provider = MSSQL';
```

```
BdpDataAdapter1.SelectCommand.CommandText := SqlStr;
```

```
BdpDataAdapter1.SelectCommand.Connection := BdpConnection1;
```

```
DataSet1.Tables.Clear;
```

```
try
```

```
    BdpDataAdapter1.Fill(DataSet1);
```

```
    DataGrid1.DataSource := DataSet1.Tables[0];
```

```
except on e: Exception do
```

```
    MessageBox.Show(e.Message);
```

```
end;  
  
if BdpConnection1 <> nil then //关闭连接  
    if BdpConnection1.State = System.Data.ConnectionState.Open  
then BdpConnection1.Close;  
end;
```

10.3.3 浏览数据

代码编写完后,编译、运行该程序,结果如图 10.4 所示。

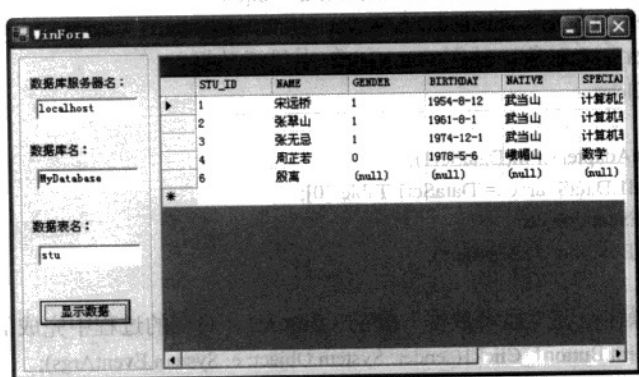


图 10.4 程序 BDPViewData.dpr 的运行界面

图 10.4 中显示的数据是采用默认的数据库服务器、数据库和数据表的设置而得到,可以更改这些值,使之显示任何一张表中的数据。

10.4 BDP 数据库的开发实例

数据管理是指数据的插入、更新和删除,这要分别运用 Insert、Update、Delete 等 SQL 语句来实现。与 ADO.NET 类似,BDP 必须由 BDPCommand 组件来构造和执行没有返回值的 SQL 语句。

在本实例中,将建立一个对学生基本信息和成绩进行管理的程序,主要功能包括输入学生基本信息,对基本信息进行更改,并且能够删除学生信息。相信在学习了这部分内容以后,读者可以非常轻松地设计与实现相应的数据查询和浏览功能。

首先,在 Delphi 2005 IDE 中创建一个 Windows Form 程序,命名为 StudentManager.dpr,并将最初生成的单元文件 Unit1.pas 改名为 MainWinForm.pas,其对应的窗体将作为应用程序的主界面。然后利用 IDE 菜单依次加入两个单元文件,分别命名为 AddWinForm.pas 和 Update_DeleteWinForm.pas,它们对应的窗体分别是数据添加界面、数据更新界面和数据删除界面,其中数据更新和删除功能将在 Update_DeleteWinForm 对应的窗体中实现。

显然,主界面窗体的作用除了作为应用程序的“门户”以外,主要是用于打开其他的两个数据管理窗体界面。由于主窗体的代码实现较为容易,故在此不再赘述。下面主要介绍三个数据管理模块的具体设计与实现问题。

10.4.1 数据插入模块的设计与实现

在单元文件 AddWinForm 对应的窗体中加入若干组件，并对它们进行布局设计，同时设置各组件的属性，如表 10.2 所示。

表 10.2 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TextBox	TextBox1	Text	(空值)
	TextBox2	Text	(空值)
	TextBox3	Text	(空值)
	TextBox4	Text	(空值)
	TextBox5	Text	(空值)
	TextBox6	Text	(空值)
	TextBox7	Text	(空值)
	TextBox8	Text	(空值)
		Multiline	True
Label	Label1	Text	学号
	Label2	Text	姓名
	Label3	Text	性别
	Label4	Text	出生日期
	Label5	Text	籍贯
	Label6	Text	专业
	Label7	Text	成绩
	Label8	Text	备 注:
GroupBox	GroupBox	Text	(空值)
Button	Button1	Text	插 入
	Button2	Text	重 置
Form	Form1	Text	利用 BDP 组件实现数据插入

接下来，分别为两个按钮编写事件处理代码。显然，主要是“插入”按钮 (Button1)，其对应的过程代码如下：

```
procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
```

```
var strSQL:string;
```

```
  n:integer;
```

```
  MyBdpConnection:BdpConnection;
```

```
  MyBDPCommand:BDPCommand;
```

```
  stu_id,name,gender,birthday,native,speciality,grade,remark :string;
```

```
begin
```

```

stu_id := TextBox1.Text;
name := TextBox2.Text;
gender := TextBox3.Text;
birthday := TextBox4.Text;
native := TextBox5.Text;
speciality := TextBox6.Text;
grade := TextBox7.Text;
remark := TextBox8.Text;
strSQL := 'delete from stu where name = "赵敏"';
strSQL := 'insert into stu values(' + stu_id + ', ' + name + ', ' + gender + ', ' + birthday + ', ' + native + ', ' +
speciality + ', ' + grade + ', ' + remark + ')';
MyBdpConnection := BdpConnection.Create('database=MyDatabase;
                                         hostname=localhost;
                                         assembly = Borland.Data.Mssql,
                                         Version=2.0.0.0,
                                         Culture=neutral,
                                         PublicKeyToken= 91d62ebb5b0d1b1b;
                                         vendorclient = sqloledb.dll;
                                         provider = MSSQL;
                                         username=sa;
                                         password=123');

MyBdpConnection.Open;
MyBDPCommand := BdpCommand.Create;
MyBDPCommand.Connection := MyBdpConnection;
MyBDPCommand.CommandType := System.Data.CommandType.Text;
MyBDPCommand.CommandText := strSQL;
try
    n:=MyBDPCommand.ExecuteNonQuery;
    MessageBox.Show('成功插入'+n.ToString+'条记录! ');
except on e: Exception do
    MessageBox.Show(e.Message);
end;
MyBdpConnection.Close;
end;

```

“重置”按钮（Button2）的功能是清除各文本框中的字符，其对应的过程代码如下：

```

procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);
begin
    TextBox1.Text := '';
    TextBox2.Text := '';
    TextBox3.Text := '';
    TextBox4.Text := '';
    TextBox5.Text := '';
    TextBox6.Text := '';
    TextBox7.Text := '';
    TextBox8.Text := '';
end;

```

代码编写完成后，运行该程序，可以从该界面上的文本框中输入数据，然后单击“插入”按钮即可完成数据的插入操作，如图 10.5 所示。



图 10.5 插入数据

10.4.2 数据更新模块的设计与实现

对于数据更新，最直观的做法是在网格中直接修改然后保存。在第 8 章中对这种更新方法已经作了介绍，但那是基于 ADO.NET 组件的数据更新方法。而对于基于 BDP 组件的这种数据更新方法也可以模仿该方法，在此不再赘述。

为了学习新技巧，本节介绍另一种方法，其基本思想是：用鼠标在网格（DataGrid 对象）中选择一条记录，使它在文本框中显示，然后在文本框中修改，最后保存修改的数据。另外约定，该程序只提供对姓名和成绩的修改功能，其他字段修改功能的实现可由此推广。

可以按照下列步骤来完成这个功能。

(1) 在单元文件 Update_DeleteWinForm 对应的窗体中，加入下列组件：DataGrid 组件一个，TextBox 组件两个，GroupBox 组件一个，Label 组件两个，Button 组件一个。

(2) 设置各组件的属性，结果如表 10.3 所示，并进行布局设计，主要调整各组件的位置和大小。

表 10.3 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DataGrid	ReadOnly	Text	(空值)
TextBox	Label1	Text	宋远桥
	Label2	Text	98
GroupBox	GroupBox	Text	更新数据
Button	Button1	Text	保存数据
Label	Label1	Text	姓名:
	Label2	Text	成绩:
Form	Form1	Text	利用 BDP 组件实现数据更新与删除

(3) 为在程序运行时自动在 DataGrid 中显示数据, 在窗体的 TWinForm_Load 方法中编写相应的代码, 结果如下:

```
procedure TWinForm.TWinForm_Load(sender: System.Object; e: System.EventArgs);
begin
  BdpConnection1 := BDPCConnection.Create('database=MyDatabase;
                                         hostname=localhost;
                                         assembly= Borland.Data.Mssql,
                                         Version=2.0.0.0,
                                         Culture=neutral,
                                         PublicKeyToken=91d62ebb5b0d1b1b;
                                         vendorclient = sqloledb.dll;
                                         provider = MSSQL;
                                         username=sa;
                                         password=123');

  BdpConnection1.Open;
  BdpDataAdapter1.SelectCommand.CommandText := 'select * from stu';
  DataSet1 := DataSet.Create;
  BdpDataAdapter1.Fill(DataSet1,'stuTable');
  DataGrid1.DataSource := DataSet1.Tables['stuTable'];
  TextBox1.Text := DataGrid1.Item[0,1].ToString;
  TextBox2.Text := DataGrid1.Item[0,6].ToString;
end;
```

(4) 实现“把当前记录放到文本框来修改”的功能, 这可以在 DataGrid 组件的 MouseUp 方法中完成, 代码如下:

```
procedure TWinForm.DataGrid1_MouseUp(sender: System.Object;
                                     e: System.Windows.Forms.MouseEventArgs);
var SqlStr:string;
    rowIndex:integer;
begin
  rowIndex := DataGrid1.CurrentRowIndex;
  cellValue := DataGrid1.Item[rowIndex,0].ToString; // cellValue 为全局变量
  TextBox1.Text := DataGrid1.Item[rowIndex,1].ToString;
  TextBox2.Text := DataGrid1.Item[rowIndex,6].ToString;
end;
```

编写完成后, 单元文件 Update_DeleteWinForm 的所有代码如下:

```
unit Update_DeleteWinForm;
```

```
interface
```

```
uses
```

```
  System.Drawing, System.Collections, System.ComponentModel,
  System.Windows.Forms, System.Data, Borland.Data.Provider,
  System.Globalization, Borland.Data.Common;
```

```
type
```

```

TWinForm = class(System.Windows.Forms.Form)
{$REGION 'Designer Managed Code'}
strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
    Components: System.ComponentModel.Container;
    DataGrid1: System.Windows.Forms.DataGrid;
    BdpConnection1: Borland.Data.Provider.BdpConnection;
    bdpSelectCommand1: Borland.Data.Provider.BdpCommand;
    bdpInsertCommand1: Borland.Data.Provider.BdpCommand;
    bdpUpdateCommand1: Borland.Data.Provider.BdpCommand;
    bdpDeleteCommand1: Borland.Data.Provider.BdpCommand;
    BdpDataAdapter1: Borland.Data.Provider.BdpDataAdapter;
    BdpCommand1: Borland.Data.Provider.BdpCommand;
    DataSet1: System.Data.DataSet;
    Button1: System.Windows.Forms.Button;
    TextBox1: System.Windows.Forms.TextBox;
    Label1: System.Windows.Forms.Label;
    Label2: System.Windows.Forms.Label;
    TextBox2: System.Windows.Forms.TextBox;
    GroupBox1: System.Windows.Forms.GroupBox;

    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
    procedure TWinForm_Load(sender: System.Object; e: System.EventArgs);
    procedure Button2_Click(sender: System.Object; e: System.EventArgs);
    procedure DataGrid1_MouseUp(sender: System.Object;
                                e: System.Windows.Forms.MouseEventHandler);

{$ENDREGION}
strict protected

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(Disposing: Boolean); override;
private
    { Private Declarations }
    {
        myConnection:SqlConnection;
        myCommand:SqlCommand;
        myDataAdapter:SqlDataAdapter;
        myDataSet:DataSet;
    }

```



```

        myDataRow:DataRow;
        mySqlParameter: SqlParameter;
    }
    cellValue:string;
public
    constructor Create;
end;
[assembly: RuntimeRequiredAttribute(TypeOf(TWinForm))]
implementation
{$AUTOBOX ON}
{$REGION 'Windows Form Designer generated code'}

/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWinForm.InitializeComponent;
begin
    Self.DataGrid1 := System.Windows.Forms.DataGrid.Create;
    Self.BdpConnection1 := Borland.Data.Provider.BdpConnection.Create;
    Self.bdpSelectCommand1 := Borland.Data.Provider.BdpCommand.Create;
    Self.bdpInsertCommand1 := Borland.Data.Provider.BdpCommand.Create;
    Self.bdpUpdateCommand1 := Borland.Data.Provider.BdpCommand.Create;
    Self.bdpDeleteCommand1 := Borland.Data.Provider.BdpCommand.Create;
    Self.BdpDataAdapter1 := Borland.Data.Provider.BdpDataAdapter.Create;
    Self.BdpCommand1 := Borland.Data.Provider.BdpCommand.Create;
    Self.DataSet1 := System.Data.DataSet.Create;
    Self.Button1 := System.Windows.Forms.Button.Create;
    Self.Label1 := System.Windows.Forms.Label.Create;
    Self.Label2 := System.Windows.Forms.Label.Create;
    Self.TextBox2 := System.Windows.Forms.TextBox.Create;
    Self.TextBox1 := System.Windows.Forms.TextBox.Create;
    Self.GroupBox1 := System.Windows.Forms.GroupBox.Create;
    (System.ComponentModel.ISupportInitialize)(Self.DataGrid1).BeginInit;
    (System.ComponentModel.ISupportInitialize)(Self.BdpDataAdapter1).BeginInit;
    (System.ComponentModel.ISupportInitialize)(Self.DataSet1).BeginInit;
    Self.GroupBox1.SuspendLayout;
    Self.SuspendLayout;

    // DataGrid1
    Self.DataGrid1.DataMember := '';
    Self.DataGrid1.HeaderForeColor := System.Drawing.SystemColors.ControlText;
    Self.DataGrid1.Location := System.Drawing.Point.Create(8, 8);
    Self.DataGrid1.Name := 'DataGrid1';
    Self.DataGrid1.ReadOnly := True;
    Self.DataGrid1.Size := System.Drawing.Size.Create(496, 216);

```

```
Self.DataGrid1.TabIndex := 1;
Include(Self.DataGrid1.MouseUp, Self.DataGrid1_MouseUp);

// BdpConnection1
Self.BdpConnection1.ConnectionOptions := 'transaction isolation=ReadCommit' +
'ted;blobsize=1024';
Self.BdpConnection1.ConnectionString := 'assembly=Borland.Data.Mssql, Vers' +
'ion=2.0.0.0, Culture=neutral, PublicKeyToken=91d62ebb5b0d1b1b;vendorclien' +
't=sqloledb.dll;osauthentication=True;database=MyDatabase;username=myuser;' +
'hostname=localhost;password=mypass;provider=MSSQL';

// bdpSelectCommand1
Self.bdpSelectCommand1.CommandOptions := nil;
Self.bdpSelectCommand1.CommandText := '';
Self.bdpSelectCommand1.CommandType := System.Data.CommandType.Text;
Self.bdpSelectCommand1.Connection := Self.BdpConnection1;
Self.bdpSelectCommand1.ParameterCount := (SmallInt(0));
Self.bdpSelectCommand1.SchemaName := nil;
Self.bdpSelectCommand1.Transaction := nil;
Self.bdpSelectCommand1.UpdatedRowSource := System.Data.UpdateRowSource.None;

// bdpInsertCommand1
Self.bdpInsertCommand1.CommandOptions := nil;
Self.bdpInsertCommand1.CommandText := '';
Self.bdpInsertCommand1.CommandType := System.Data.CommandType.Text;
Self.bdpInsertCommand1.Connection := nil;
Self.bdpInsertCommand1.ParameterCount := (SmallInt(0));
Self.bdpInsertCommand1.SchemaName := nil;
Self.bdpInsertCommand1.Transaction := nil;
Self.bdpInsertCommand1.UpdatedRowSource := System.Data.UpdateRowSource.None;

// bdpUpdateCommand1
Self.bdpUpdateCommand1.CommandOptions := nil;
Self.bdpUpdateCommand1.CommandText := '';
Self.bdpUpdateCommand1.CommandType := System.Data.CommandType.Text;
Self.bdpUpdateCommand1.Connection := nil;
Self.bdpUpdateCommand1.ParameterCount := (SmallInt(0));
Self.bdpUpdateCommand1.SchemaName := nil;
Self.bdpUpdateCommand1.Transaction := nil;
Self.bdpUpdateCommand1.UpdatedRowSource := System.Data.UpdateRowSource.Both;

// bdpDeleteCommand1
Self.bdpDeleteCommand1.CommandOptions := nil;
Self.bdpDeleteCommand1.CommandText := '';
Self.bdpDeleteCommand1.CommandType := System.Data.CommandType.Text;
Self.bdpDeleteCommand1.Connection := nil;
```

```
Self.bdpDeleteCommand1.ParameterCount := (SmallInt(0));
Self.bdpDeleteCommand1.SchemaName := nil;
Self.bdpDeleteCommand1.Transaction := nil;
Self.bdpDeleteCommand1.UpdatedRowSource := System.Data.UpdateRowSource.None;

// BdpDataAdapter1
Self.BdpDataAdapter1.Active := False;
Self.BdpDataAdapter1.DataSet := nil;
Self.BdpDataAdapter1.DataTable := nil;
Self.BdpDataAdapter1.DeleteCommand := Self.bdpDeleteCommand1;
Self.BdpDataAdapter1.InsertCommand := Self.bdpInsertCommand1;
Self.BdpDataAdapter1.SelectCommand := Self.bdpSelectCommand1;
Self.BdpDataAdapter1.StartRecord := 0;
Self.BdpDataAdapter1.UpdateCommand := Self.bdpUpdateCommand1;

// BdpCommand1
Self.BdpCommand1.CommandOptions := nil;
Self.BdpCommand1.CommandText := nil;
Self.BdpCommand1.CommandType := System.Data.CommandType.Text;
Self.BdpCommand1.Connection := nil;
Self.BdpCommand1.ParameterCount := (SmallInt(0));
Self.BdpCommand1.SchemaName := nil;
Self.BdpCommand1.Transaction := nil;
Self.BdpCommand1.UpdatedRowSource := System.Data.UpdateRowSource.None;

// DataSet1
Self.DataSet1.DataSetName := 'NewDataSet';
Self.DataSet1.Locale := System.Globalization.CultureInfo.Create('zh-CN');

// Button1
Self.Button1.Location := System.Drawing.Point.Create(48, 123);
Self.Button1.Name := 'Button1';
Self.Button1.TabIndex := 2;
Self.Button1.Text := '保存数据';
Include(Self.Button1.Click, Self.Button2_Click);

// Label1
Self.Label1.Location := System.Drawing.Point.Create(8, 40);
Self.Label1.Name := 'Label1';
Self.Label1.TabIndex := 4;
Self.Label1.Text := '姓名: ';

// Label2
Self.Label2.Location := System.Drawing.Point.Create(8, 88);
Self.Label2.Name := 'Label2';
Self.Label2.Size := System.Drawing.Size.Create(48, 23);
```

```

Self.Label2.TabIndex := 5;
Self.Label2.Text := '成绩: ';

// TextBox2
Self.TextBox2.Location := System.Drawing.Point.Create(61, 85);
Self.TextBox2.Name := 'TextBox2';
Self.TextBox2.TabIndex := 6;
Self.TextBox2.Text := '';

// TextBox1
Self.TextBox1.Location := System.Drawing.Point.Create(61, 37);
Self.TextBox1.Name := 'TextBox1';
Self.TextBox1.TabIndex := 7;
Self.TextBox1.Text := '';

// GroupBox1
Self.GroupBox1.Controls.Add(Self.Label2);
Self.GroupBox1.Controls.Add(Self.TextBox2);
Self.GroupBox1.Controls.Add(Self.TextBox1);
Self.GroupBox1.Controls.Add(Self.Button1);
Self.GroupBox1.Controls.Add(Self.Label1);
Self.GroupBox1.Location := System.Drawing.Point.Create(8, 240);
Self.GroupBox1.Name := 'GroupBox1';
Self.GroupBox1.Size := System.Drawing.Size.Create(216, 160);
Self.GroupBox1.TabIndex := 8;
Self.GroupBox1.TabStop := False;
Self.GroupBox1.Text := '更新数据';

// TWinForm
Self.AutoScaleBaseSize := System.Drawing.Size.Create(6, 14);
Self.ClientSize := System.Drawing.Size.Create(520, 406);
Self.Controls.Add(Self.GroupBox1);
Self.Controls.Add(Self.DataGrid1);
Self.Name := 'TWinForm';
Self.Text := '利用 BDP 组件实现数据更新与删除';
Include(Self.Load, Self.TWinForm_Load);
(System.ComponentModel.ISupportInitialize.SupportInitialize(Self.DataGrid1)).EndInit;
(System.ComponentModel.ISupportInitialize.SupportInitialize(Self.BdpDataAdapter1)).EndInit;
(System.ComponentModel.ISupportInitialize.SupportInitialize(Self.DataSet1)).EndInit;
Self.GroupBox1.ResumeLayout(False);
Self.ResumeLayout(False);
end;
{$ENDREGION}

procedure TWinForm.Dispose(Disposing: Boolean);
begin

```

```

    if Disposing then
    begin
        if Components <> nil then
            Components.Dispose();
        end;
        inherited Dispose(Disposing);
    end;

constructor TWinForm.Create;
begin
    inherited Create;
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent;
    //
    // TODO: Add any constructor code after InitializeComponent call
    //
end;

procedure TWinForm.DataGrid1_MouseUp(sender: System.Object;
                                     e: System.Windows.Forms.MouseEventArgs);
var SqlStr:string;
    rowIndex:integer;
begin
    rowIndex := DataGrid1.CurrentRowIndex;
    cellValue := DataGrid1.Item[rowIndex,0].ToString;
    TextBox1.Text := DataGrid1.Item[rowIndex,1].ToString;
    TextBox2.Text := DataGrid1.Item[rowIndex,6].ToString;
end;

procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);
var SqlStr:string;
begin
    BdpCommand1 := BdpConnection1.CreateCommand;
    SqlStr:= 'update stu set name = '''+ TextBox1.Text +'',
grade = '''+ TextBox2.Text +'' where stu_id = '+ cellValue;
    BdpCommand1.CommandText := SqlStr;
    BdpCommand1.ExecuteNonQuery;
    DataSet1.Tables.clear;
    BdpDataAdapter1.Fill(DataSet1,'stuTable');
    DataGrid1.DataSource := DataSet1.Tables.Item['stuTable'];
end;

procedure TWinForm.TWinForm_Load(sender: System.Object; e: System.EventArgs);
begin

```

```

BdpConnection1 := BDPConnection.Create('database= MyDatabase;
hostname= localhost;
assembly= Borland.Data.Mssql,
Version=2.0.0.0,
Culture=neutral,
PublicKeyToken = 91d62ebb5b0d1b1b;
vendorclient = sqloledb.dll;
provider=MSSQL;
username=sa;
password=123');

```

```

BdpConnection1.Open;
BdpDataAdapter1.SelectCommand.CommandText := 'select * from stu';
DataSet1 :=DataSet.Create;
BdpDataAdapter1.Fill(DataSet1,'stuTable');
DataGrid1.DataSource := DataSet1.Tables['stuTable'];
TextBox1.Text := DataGrid1.Item[0,1].ToString;
TextBox2.Text := DataGrid1.Item[0,6].ToString;
end;

```

end.

(5) 运行该程序, 其运行界面如图 10.6 所示。在该界面中, 可以根据需要修改任一记录的 NAME 和 GRADE 字段值, 读者也可以由此推广出修改其他字段值的实现方法。

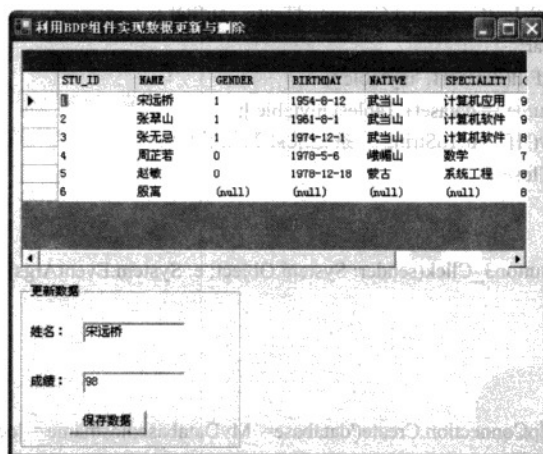


图 10.6 程序 BDPUpdDel.dpr 的运行界面

10.4.3 数据删除模块的设计与实现

在此, 按照两种方式删除记录: 一种是删除当前记录, 另一种是删除所有记录, 它们都是用 SQL 语句 Delete 来实现的。

数据删除功能与数据更新功能都是在单元文件 Update_DeleteWinForm 对应的窗体中实现的。为此在该窗体中加入一个 GroupBox 组件和两个 Button 组件, 然后把 GroupBox 组件的 Text

属性值设置为“删除数据”，两个 Button 组件的 Text 属性值分别设置为“删除当前记录”（对应 Button2）和“删除所有记录”（对应 Button3）。编写两个按钮事件的处理代码，得到的过程代码如下：

```

procedure TForm.Button2_Click(sender: System.Object; e: System.EventArgs); // Button1
var n:integer;
    strSQL:string;
begin
    BdpConnection1 := BdpConnection.Create('database= MyDatabase;hostname= localhost;assembly= Borland.
Data.Mssql, Version=2.0.0.0, Culture=neutral, PublicKeyToken= 91d62ebb5b0d1b1b; vendorclient = sqloledb.dll;
provider=MSSQL;username=sa;password=123');
    BdpConnection1.Open;
    strSQL := 'delete from stu where  stu_id = '+ cellValue;
    BDPCommand1 := BdpCommand.Create;
    BdpCommand1.Connection := BdpConnection1;
    BdpCommand1.CommandType := System.Data.CommandType.Text;
    BDPCommand1.CommandText := strSQL;
    try
        n:=BDPCommand1.ExecuteNonQuery;
    except on e: exception do
        MessageBox.Show(e.Message)
    end;
    strSQL := 'select * from stu';
    BdpDataAdapter1.SelectCommand.CommandText := strSQL;
    dataset1.tables.Clear;
    BdpDataAdapter1.Fill(dataset1, 'mytable');
    DataGrid1.DataSource := dataset1.Tables['mytable'];
    MessageBox.Show('有 '+n.ToString+' 条记录被删除了! ');
    BdpConnection1.Close;
end;

procedure TForm.Button3_Click(sender: System.Object; e: System.EventArgs);
var n:integer;
    strSQL:string;

begin
    BdpConnection1 := BdpConnection.Create('database= MyDatabase;hostname= localhost;assembly= Borland.
Data.Mssql, Version=2.0.0.0, Culture=neutral, PublicKeyToken= 91d62ebb5b0d1b1b; vendorclient = sqloledb.dll;
provider=MSSQL;username=sa;password=123');
    BdpConnection1.Open;
    strSQL := 'delete from stu';
    BDPCommand1 := BdpCommand.Create;
    BdpCommand1.Connection := BdpConnection1;
    BdpCommand1.CommandType := System.Data.CommandType.Text;
    BDPCommand1.CommandText := strSQL;
    try
        n:=BDPCommand1.ExecuteNonQuery;

```

```
except on e: exception do
    MessageBox.Show(e.Message)
end;
strSQL := 'select * from stu';
BdpDataAdapter1.SelectCommand.CommandText := strSQL;
dataset1.tables.Clear;
BdpDataAdapter1.Fill(dataset1, 'mytable');
DataGrid1.DataSource := dataset1.Tables['mytable'];
MessageBox.Show('有 '+n.ToString+ ' 条记录被删除了! ');
BdpConnection1.Close;
end;
```

修改后的窗体界面在运行后如图 10.7 所示。在该界面中, 可以用鼠标选择某一条记录, 然后单击“删除当前记录”按钮删除该条被选中的记录, 或者单击“删除所有记录”按钮删除所有的记录。

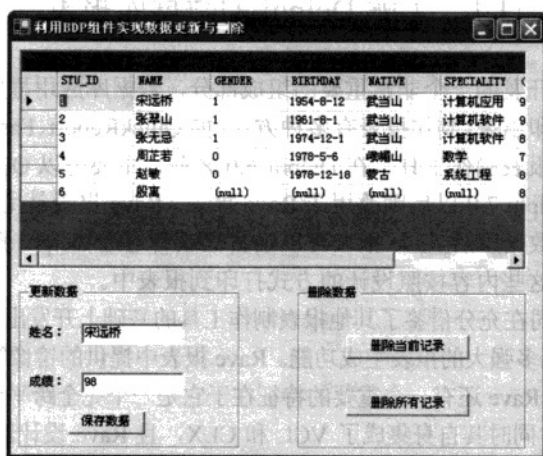


图 10.7 程序 BDPUdpDel.dpr 的运行界面 (增加删除功能)

10.5 小结

本章继续介绍 Delphi 2005 在数据库应用程序开发方面的知识, 主要是基于 BDP 的数据库应用开发技术。通过对本章的学习, 应达到以下的目标:

- 了解 BDP 数据库的特点。
- 掌握常用 BDP 组件的基本属性和方法。
- 掌握基于 BDP 的数据浏览技术。
- 熟练掌握基于 BDP 的数据管理, 包括数据插入、数据更新和数据删除等, 深入领会 DbpCommand 组件的运用。

第 11 章 熟悉 Delphi 中报表设计和数据输出

Delphi 2005 对报表设计提供了很好的支持,它把 Nevrona 公司开发的 Rave 组件嵌入其中,使得报表设计变得简便而快捷。本章主要介绍基于 Rave 组件的报表设计技术,涉及的内容要点包括:

- 常用 Rave 组件的有关属性和方法,包括 RvDataSetConnection、RvProject、RvSystem 和 RvNDRWriter 等组件。
- 基于 Rave 组件的动态报表设计技术。
- 报表设计和开发的一般方法。

11.1 了解 Delphi 的数据库报表

报表设计是数据库开发的一个非常重要的组成部分,数据库应用程序通常都要生成报表,以把相应的输出结果打印出来。制作报表有多种方法,如 QuickReport、FastReport、ReportBuilder 等组件都是非常优秀的报表制作工具。在 Delphi 6.0 之前,主要是以 QuickReport 作为可视化报表的设计方案,从 Delphi 7.0 以后则改用了 Rave 报表。Rave 报表是用 Delphi 数据来制作报表的,在设计了 Delphi 数据库后 Rave 报表可以由自己或者通过 Delphi 直接连接到数据库,获取数据库中的内容并把这些内容按照设计的方式打印到报表中。

Rave 是 Nevrona 公司在充分借鉴了其他报表制作工具的基础上开发出的一种简便、易学的报表制作工具,它提供了很多强大的报表生成功能。Rave 报表中提供的镜像等技术可很好地实现代码的重用和报表的维护。Rave 还有一个重要的特征在于它是一个完全跨平台的解决方案,可以在多种不同的平台下运行,同时其自身集成了 VCL 和 CLX,且 Rave 设计器本身也是一个以 CLX 编写的跨平台的应用程序。这也是 Borland 公司看中 Rave 的主要原因。实际上,Rave 与 Delphi 并无必然的联系,Rave 是个独立的程序,可以单独设计和执行报表,也可以独立连接数据库来打印数据报表,当然在 Delphi 应用程序开发中更多是挂在 Delphi 程序下,由 Delphi 应用程序执行报表的打印功能。Rave 报表、Delphi 应用程序和数据库的关系可以用图 11.1 来表示。

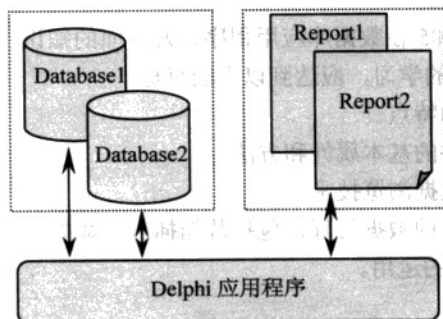


图 11.1 Rave 报表、Delphi 应用程序和数据库的关系

在 Delphi 2005 中已经默认安装了 Rave 组件, 该组件具有以下几个特点:

- 简单、易学、易用, 模块化强。
- 具有功能强大的报表设计器, 可设计出非常漂亮的数据报表。
- 可以独立于 Delphi 进行设计, 成品的报表能够与 Delphi 应用程序高度无缝集成。
- 具备优秀的数据库连接能力。

在 Delphi 2005 中, Rave 组件位于工具面板中的 Rave 标签页上, 一共包含 12 个组件。下一节将对一些常用的组件进行介绍。

11.2 制作 Delphi 的第一个数据库报表

为获得对报表制作的一个感性认识, 先通过使用 Rave 组件生成一个简单的报表。这个报表用于浏览 Access 数据库 ResultDB.mdb 中数据表 student 的数据(该数据库的创建见 7.5.2 节)。

创建步骤如下:

(1) 首先创建一个 VCL Form 应用程序, 并在窗体上加入下列组件: Table、RvDataSet-Connection、RvProject 和 Button 组件各一个。然后把 Table 组件的 DatabaseName 属性值设置为 AccessDSN (AccessDSN 是连接数据库 ResultDB.mdb 的数据源), TableName 属性值设置为 student; RvDataSetConnection 组件的 DataSet 属性值设置为 Table1 (Table 组件的 name 属性值)。

(2) 在 Delphi 2005 IDE 中, 选择菜单 Tools→Rave Reports Designer 命令, 启动 Rave Reports 6.0 (简称报表设计器), 如图 11.2 所示。

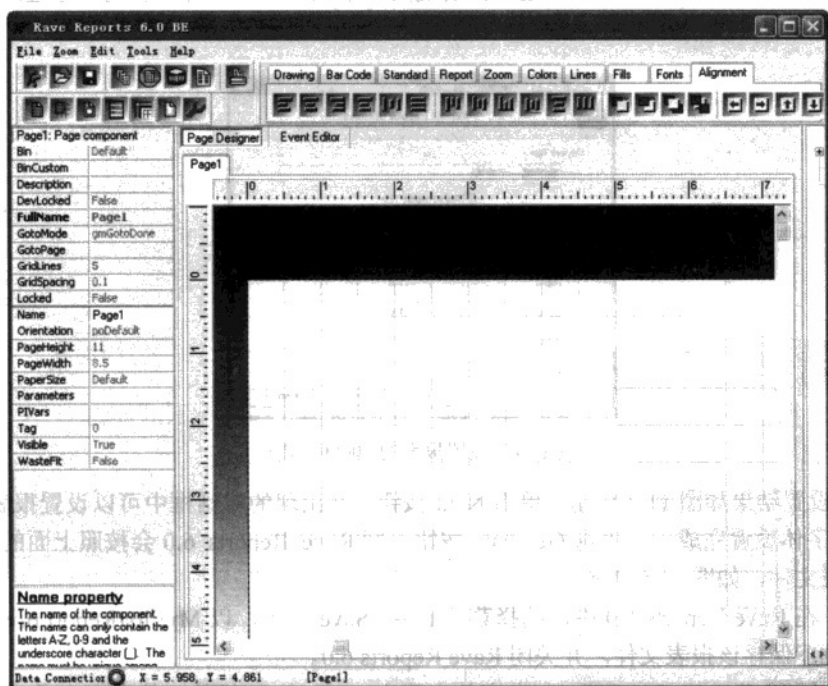


图 11.2 报表设计器

(3) 在 Rave Reports 6.0 中, 选择菜单 File→New Data Object 命令, 打开 Data Connections 对话框, 如图 11.3 所示。

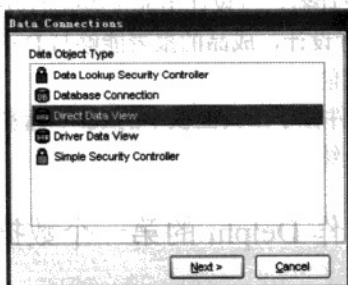


图 11.3 Data Connections 对话框

(4) 在 Data Connections 对话框中选择 Direct Data View 选项, 然后单击 Next 按钮, 在弹出的对话框中选择已经被激活的数据集连接 RvDataSetConnection1, 然后单击 Finish 按钮。

(5) 上述操作完成以后, 在 Rave Reports 6.0 中, 选择菜单 Tools→Report Wizards→Simple Table 命令, 打开 Simple Table 对话框。

(6) 在 Simple Table 对话框中选择 DataView1, 单击 Next 按钮, 在出现的对话框中选择所有的项, 表示要在报表中显示所有的字段。

(7) 单击 Next 按钮, 继续出现 Simple Table 对话框, 以用于设置报表中各字段的显示顺序。在此, 用默认值, 单击 Next 按钮, 在出现的对话框中可以设置报表的一些边距及其报表标题等, 如图 11.4 所示。

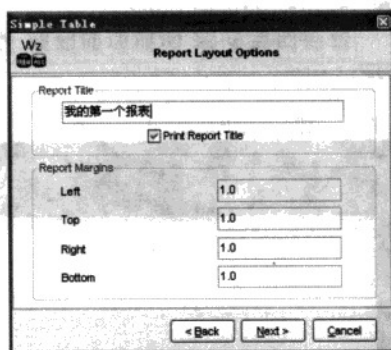



图 11.4 设置报表的边距和标题

(8) 设置结果如图 11.4 所示, 单击 Next 按钮, 在出现的对话框中可以设置报表的字体。

(9) 字体设置完成后, 单击 Generate 按钮, 则 Rave Reports 6.0 会按照上面的设置生成相应的报表文件, 如图 11.5 所示。

(10) 在 Rave Reports 6.0 中, 选择菜单 File→Save 命令, 以 MyFirstReport.rav 为名称在某个目录下保存该报表文件, 并关闭 Rave Reports 6.0。

(11) 回到 Delphi 2005 IDE, 在窗体中选中 RvProject 组件, 然后在对象观察器中把 RvProject 组件的 ProjectFile 属性值设置为刚创建的报表文件名, 方法是: 找到 ProjectFile 属

性项并单击其右边的省略号按钮 ，在弹出的 Select Rave Project File 对话框中找到 MyFirstReport.rav 文件所在的目录并选择它，然后单击“打开”按钮，ProjectFile 属性值设置完毕。

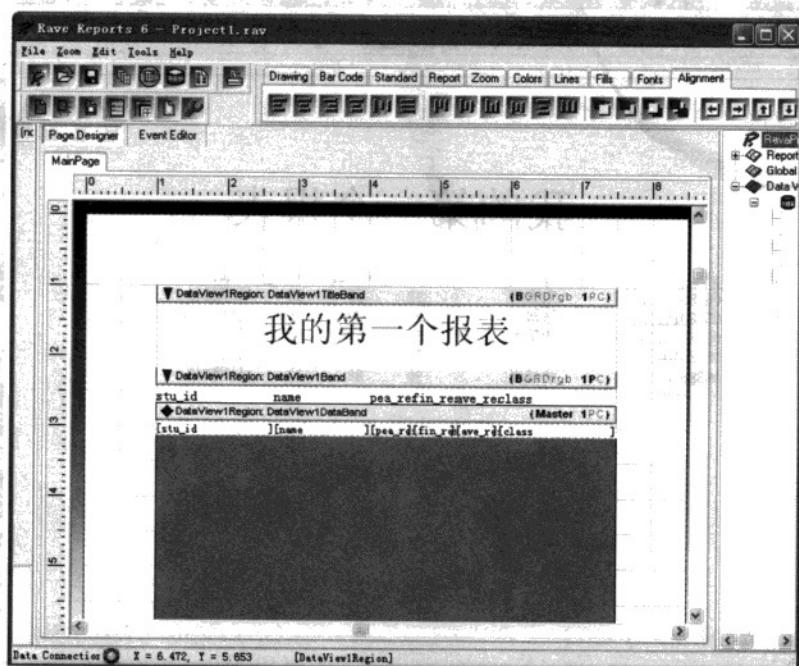


图 11.5 生成的报表（设计阶段）

(12) 为打开报表，编写 Button 组件的事件处理代码，并把其 Caption 属性值设置为“打开报表”。结果的过程代码如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    rvproject1.Execute;
end;
```

(13) 运行该 VCL Form 应用程序，然后在运行界面上单击“打开报表”按钮，则出现 Output Options 对话框，如图 11.6 所示。

在 Output Options 对话框中，可以选择三种输出方式：一种是输出到打印机（打印），另一种是输出到显示器（浏览用），还有一种是输出到一个文件保存起来。

(14) 在此，选择 Preview 单选框（只在显示器上浏览），然后单击 OK 按钮，将出现如图 11.7 所示的数据库报表浏览界面。

至此，一个简单的数据库报表就已经生成了。这个报表主要运用两个 Rave 组件，即 RvDataSetConnection 和 RvProject。实际上，Rave 组件包含

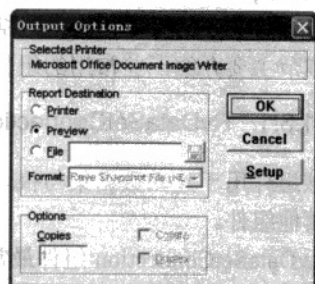


图 11.6 Output Options 对话框

了许多功能强大的组件,了解这些组件的属性和方法对高效地设计一个“漂亮”的数据库报表是至关重要的。

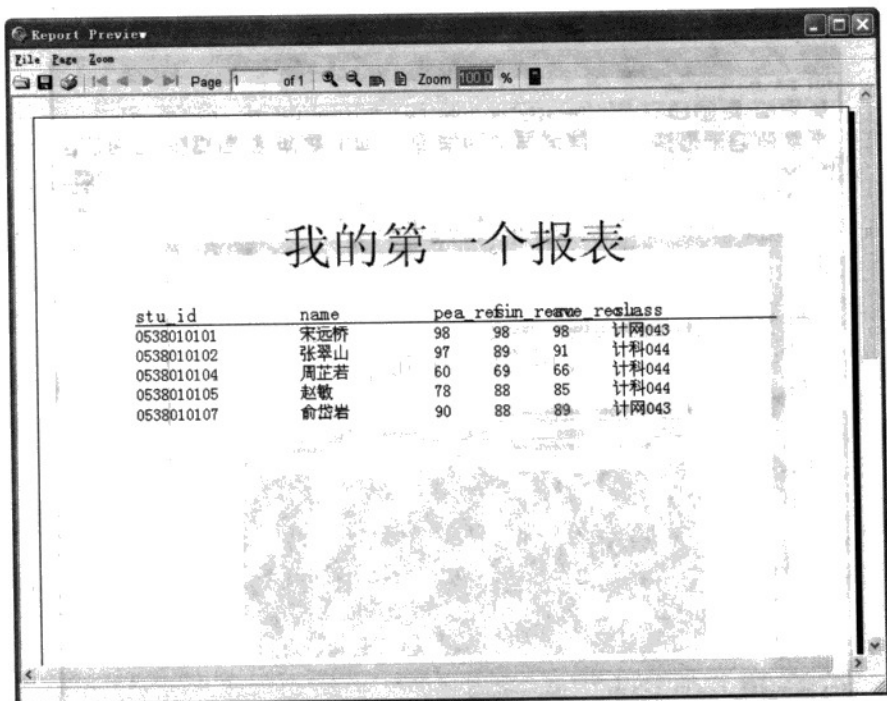


图 11.7 数据库报表浏览界面

在报表设计器中设计报表时,随时都可以选择菜单 **File→Execute Report** 命令或者按 **F9** 功能键来观察当前报表的设计效果。


11.3 熟悉常用的 Rave 组件

在 Delphi 2005 中, Rave 组件一共包含 12 个组件,它们位于工具面板中的 Rave 标签页上,用于实现对报表的操作,提供包括与报表之间的数据连接和生成报表的格式操纵等功能。

本节主要介绍一些常用的组件。

11.3.1 RvDataSetConnection 组件

该组件用于建立数据集与报表之间的连接,数据集可以是 **DataSet** 或者由 **DataSet** 派生而来的其他组件。

RvDataSetConnection 组件最常用到的属性是 **DataSet**,它用于设置要连接的数据集对象,当单击该属性项右边的下拉列表按钮  时,当前可获得的数据集对象将在下拉列表框中列出,如图 11.8 所示,从中选择相应的数据集对象即可。

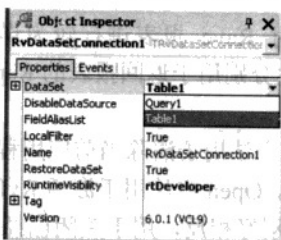


图 11.8 设置 RvDataSetConnection 组件的 DataSet 属性值

其 FieldAliasList 属性可以用来修改数据集的字段名, 将其替换为容易理解和记忆的名称。OnSetSort 和 OnSetFilter 事件则分别用于在 Rave 报表中对查找项或主/详关系进行分类或过滤。

11.3.2 RvProject 组件

在 Rave 报表设计中, 这个组件是最为重要的一个, 也是使用频率最高的一个组件。程序员可以通过这个报表完成报表的打印、文件的生成、输出, 也可以通过它来点用相应的报表设计器。RvProject 组件要设置的属性也不多, 从上一节也可以看出, 主要是使用它的 ProjectFile 属性。该属性用于指定要连接的报表文件 (Rave 项目文件), 这是一种以 rav 为扩展名的文件。RvProject 组件的作用就是, 当激活它时, .rav 文件将被装入内存以用于显示或者打印。如果要关闭 RvProject 对象时, 可调用它的 Closed 方法, 并使之释放所占用的系统资源。RvProject 组件其他常用的属性、方法及其含义说明如下:

(1) Active 属性。用于激活或者关闭 RvProject 对象, 当 Active 属性被赋值为 true 时, RvProject 被激活, 相当于执行其 Open 方法; 如果赋值为 false 时, RvProject 被关闭, 相当于执行其 Closed 方法。

(2) DLLFile 属性。指定发行报表时需要的 dll 文件, 这时 LoadDesigner 属性值应为 true。

(3) Engine 属性。指定相应报表生成的目的地。它的数据类型为 TRpComponent, 默认值是 RvSystem 对象。该对象用于打印、打印预览、生成打印文件等。如果要使数据库输出的结果为 RTF、HTML、PDF、TXT 中的一种, 那么需要使该属性值为已经创建的 RvNDRWriter 对象。

(4) LoadDesigner 属性。用于确定终端用户是否可以调用报表设计器。如果 LoadDesigner 属性的值为 true, 那么终端用户就可以调用报表设计器; 如果它的值为 false, 那么最终用户就没有权力调用报表设计器。

(5) ProjectFile 属性。指定要调用的报表项目文件, 指定详细目录路径, 当 TRvProject 被打开时载入。

(6) StoreRAV 属性。如果要使报表嵌入到 Delphi 生成的 exe 文件中, 则用 StoreRAV 属性指定相应的报表项目文件。其返回值是布尔类型, 可在运行时调用。当指定的报表项目文件是可执行的则返回 true, 否则返回 false。该属性返回值说明了一个报表项目文件是可执行的还是不可执行的。

(7) ReportName、ReportFullName 和 ReportDesc 属性。一个 Rave 报表一般由三种方式来描述: name (名字)、full name (全名) 和 description (描述)。name 是报表的唯一标识符, 要满足一般对象 name 属性值的命名规则; full name 是 Rave 报表的概要说明字符串, 类似于

小标题的功能；description 用于对 Rave 报表进行较为详细的描述。显然，ReportName、ReportFullName 和 ReportDesc 属性分别用于返回报表的 ReportName、full name 和 description 信息。

(8) ClearRaveBlob 方法。该方法用于清除当前应用程序载入的报表项目。

(9) Open 方法和 Close 方法。Open 方法用于激活 RvProject 组件，以把相应的报表项目载入到内存中，Close 方法的作用刚好相反，用于关闭已经处于激活状态的 RvProject 组件，使之释放内存资源。

(10) Design 方法。用于启动 Rave 报表的可视化设计器，以对当前报表进行设计。

(11) DesignReport 方法。该方法具有 Design 方法的功能，但不同的是它启动的设计器还可以用来设计除当前报表以外的其他由属性 ReportName 指定的报表。

(12) SelectReport 方法。用于选择相应的报表，如果选择成功，该方法将返回 true，否则返回 false。

(13) Execute 方法。执行（包括打印和预览等）被 SelectReport 选择的报表文件。

(14) ExecuteReport 方法。用于执行指定的报表项目，其调用格式为：

```
RvProject1.ExecuteReport(ReportName);
```

其中，参数 ReportName 是 string 型，是相应的报表名称。

(15) Save 方法。用于保存报表项目文件。

11.3.3 RvSystem 组件

RvSystem 组件功能非常强大，它将打印预览、便携式打印机接口和报表编写等功能集于一体，是 TRvNDRPreview、TRvNDRWriter 和 TRvNDRPrinter 组件功能的集成者，可以有效地将从 RvProject 组件接收过来的报表发送到打印机或者显示器浏览。

RvSystem 组件的主要属性和方法说明如下：

(1) DefaultDest 属性。指定输出的方式，其取值有三种，即 rdFile、rdPreview 和 rdPrinter，分别表示输出到文件、显示器（浏览）和打印机。

(2) RulerType 属性。用于指定在浏览时产生的标尺类型。其取值有七个，其含义如表 11.1 所示。

表 11.1 RulerType 的属性值及其含义

RulerType 属性值	含义
rtNone	没有标尺
rtHorizCm	使用水平方向上（横向）的标尺，单位为厘米
rtVertCm	使用垂直方向上（纵向）的标尺，单位为厘米
rtBothCm	水平方向上和垂直方向上的标尺都使用，单位为厘米
rtHorizIn	使用水平方向上（横向）的标尺，单位为英寸
rtVertIn	使用垂直方向上（纵向）的标尺，单位为英寸
rtBothIn	水平方向上和垂直方向上的标尺都使用，单位为英寸

(3) SystemFiler 属性。当 DefaultDest 属性值设置为 rbFile 时，SystemFiler 属性用于指定

报表输出达到的目的文件。

(4) `SystemOptions` 属性。这是集合类型属性，用于设置报表输出属性。例如，可以用下列格式设置该属性：

```
RvSystem1.SystemOptions := [soWaitForOK,soAllowPrintFromPreview,soNoGenerate];
```

集合的元素及其含义如表 11.2 所示。

表 11.2 `SystemFiler` 属性值（集合）的元素含义

RulerType 属性值	含义
<code>soUseFiler</code>	把报表送到报表文件
<code>soWaitForOK</code>	在报表生成以后，要单击 OK 按钮才可以输出
<code>soShowStatus</code>	在生成报表的过程中显示屏幕状态
<code>soAllowPrintFromPreview</code>	允许在预览屏幕中打印报表
<code>soPreviewModel</code>	把预览屏幕设置为模态
<code>soNoGenerate</code>	不生成报表而直接送到打印机或者屏幕

(5) `SystemPreview` 属性。当 `DefaultDest` 属性值设置为 `rdPreview`，则需要设置这里的属性值。例如，将 `SystemPreview` 的 `FormStatus` 属性设置为 `wsMaximized`，则报表预览的窗体最大化显示。

(6) `SystemPrinter` 属性。当 `DefaultDest` 属性值设置为 `rdPrinter`，则需要设置这里的属性值。例如，将 `SystemPrinter` 的 `Orientation` 属性设置为 `poLandScape`，则报表显示为横向的。

(7) `SystemSetup` 属性。`SystemSetup` 属性有许多子属性项可以设置。例如，如果使得 `SystemSetup` 属性值（集合型）包含 `ssAllowSetup`，那么在执行报表文件时不会先出现 `Output Options` 对话框。

11.4 基于 Delphi Rave 组件的动态报表设计

本章 11.2 节中已经简单地介绍了报表的设计方法，并已经生成了一个实实在在的报表，初学者可以从中获得非常直观的感性认识。但是对于工程开发者来说，仅限于这一点设计知识是不够的。通过 11.3 节对一些常用 Rave 组件的学习，已经对它们的属性和方法有所了解，在实际开发中完全可以利用它们来提高报表设计的灵活性。为此，本节将主要介绍基于 Rave 组件编程的动态报表设计方法，一方面使读者对报表设计的基本原理有更进一步的了解，另一方面也可以有效地提高读者的报表设计水平。

11.4.1 使用 `RvSystem` 组件动态生成报表

`RvSystem` 组件的功能非常强大，它可以控制报表的显示格式。使用 `RvSystem` 组件的方法非常简单，只要把 `RvProject` 组件的 `Engine` 属性值设置为 `RvSystem` 对象即可。例如：

```
RvProject1.Engine:=RvSystem1;
```

然后，设置 `RvSystem` 对象的各种属性，以达到控制报表显示格式的目的。下面通过一个实例说明这个问题。

在 Delphi 2005 IDE 中创建 VCL Form 应用程序，命名为 `RvSystemTest.dpr`。然后在程序的

窗体中添加下列组件: Table、RvDataSetConnection、RvProject、RvSystem 和 Button 组件各一个, 并把 Table 组件的 DatabaseName 属性值和 TableName 属性值分别设为 AccessDSN (AccessDSN 是 Access 数据库 ResultDB.mdb 的数据源) 和 student, 把 RvDataSetConnection 组件的 DataSet 属性值设为 Table1 (在设计报表时需要一个被实例化的 RvDataSetConnection, 以能从中读出数据), 而对其他属性值不作设置, 由程序代码来设置。

在 Delphi 2005 IDE 中, 选择菜单 Tools→Rave Reports Designer 命令, 启动报表设计器。在报表设计器中, 选择菜单 File→New Data Object 命令, 打开 Data Connection 对话框。在该对话框中选择 Direct DataView 选项, 然后单击 Next 按钮, 在弹出的对话框中选择已经被激活的数据集连接 “RvDataSetConnection1”, 然后单击 Finish 按钮, 这样数据就被导入到报表中。

然后, 在报表设计器中, 选择菜单 Tools→Report Wizards→Simple Table 命令, 打开 Simple Table 对话框。在该对话框中选择 DataView1, 单击 Next 按钮, 在出现的对话框中选择所有的项, 表示要在报表中显示所有的字段。单击 Next 按钮, 继续出现 Simple Table 对话框, 以用于设置报表中各字段的显示顺序。在此用默认值, 单击 Next 按钮, 在出现的对话框中可以设置报表的一些边距及其报表标题等, 在这里采用默认值。接着, 单击 Next 按钮, 在出现的对话框中可以设置报表的字体, 在此也采用默认值, 单击 Generate 按钮, 则报表设计器会自动生成相应的报表文件。

在 11.2 节介绍例子中, 报表中每一列都是用列名来说明的, 而列名多是英文字母组成的字符串, 这不方便人们对报表的理解。所以, 一般要把它改为具有一定意义的词语 (注意, 这并不改变字段名)。为此, 在 DataView1DataBand 栏中, 选择 “stu_id”, 然后在报表设计器左边的属性设置方框中把 Text 的属性值由默认的 “stu_id” 改为 “学号”。用同样的方法修改其他列, 结果如图 11.9 所示。

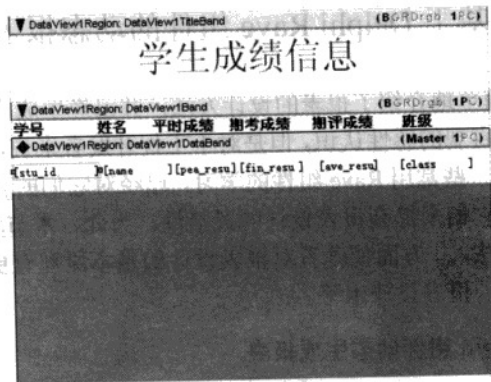



图 11.9 修改列的标题名

按 F9 功能键可以看到如图 11.10 所示的设计效果。

从修改列标题名的这个例子中可以看出, 在报表基本框架生成以后, 可以利用报表设计器左边的属性设置框等对报表作进一步的可视化设计, 直至达到用户的要求为止, 而且提供的修改功能非常强大, 程序员可以多试试, 举一反三。

学生成绩信息					
学号	姓名	平时成绩	期考成绩	期评成绩	班级
0538010101	宋远桥	98	98	98	计网043
0538010102	张翠山	97	89	91	计科044
0538010104	周芷若	60	69	66	计科044
0538010105	赵敏	78	88	85	计科044
0538010107	俞岱岩	90	88	89	计网043

图 11.10 报表的设计效果（修改列标题名后）

如果数据有多页时，报表标题“学生成绩信息”只在第一页中显示。但有的用户希望在每一页都能显示报表标题，那么如何满足这一要求呢？方法很简单：选中 DataView1TitleBand 栏，在属性设置框中找到 BandStyle 属性项并单击其右边的省略号按钮，在弹出的 Band Style Editor 对话框中单击选中 New Page 复选框，如图 11.11 所示。这样在预览时会发现，生成报表的每一页都有标题。

在报表设计完毕并保存以后，就可以调用它来显示数据了。调用报表的过程实际上就是把报表数据加载到内存的过程，并最终把报表数据显示到指定的设备上。在显示时，可以利用 RvSystem 组件来控制显示风格。作为一个例子，在 Button 组件的 Click 事件对应的过程中编写下列代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
    RvDataSetConnection1.DataSet := Table1;
```

```
    RvProject1.ProjectFile := 'C:\Documents and Settings\mengzuqiang\My Documents\Borland Studio
Projects\ch11\MyProject1.rav'; //指向报表文件
```

```
    RvProject1.Engine:=RvSystem1; //运用 RvSystem 组件
```

```
    RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
```

```
    //去掉 ssAllowSetup，使之不出现 Output Options 对话框
```

```
    RvSystem1.SystemPreview.FormHeight := 800; //使报表窗口的高度为 800 像素
```

```
    RvSystem1.SystemPreview.FormWidth := 1024;
```

```
    //使报表窗口的宽度为 1024 像素，这样使报表窗口刚好可以占满整个屏幕
```

```
    RvSystem1.SystemPreview.RulerType := rtBothCm;
```

```
    //使用水平方向上和垂直方向上的标尺，单位为厘米
```

```
    RvSystem1.SystemPreview.MarginPercent := 3; //设置报表页面距预览窗体的边距
```

```
    RvSystem1.SystemPrinter.Orientation := poLandscape;
```

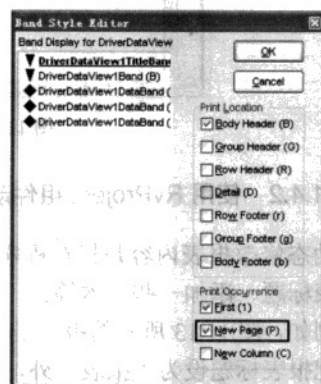


图 11.11 Band Style Editor 对话框

//设置显示页面为横向，这对于显示字段数量比较多的情况非常有效的

RvProject1.Execute;

end;

在运行程序 RvSystemTest 后，单击相应的 Button 按钮，执行指定的报表项目，结果如图 11.12 所示。

图 11.12 运用 RvSystem 控制报表显示

11.4.2 使用 RvProject 组件动态修改报表内容

动态修改报表内容是指在程序运行时通过运行代码来修改和添加报表中的有关项，如修改报表标题、添加一些文本等。

例如，图 11.13 所示的报表是采用类似 11.4.1 节介绍的方法，不同的是在报表创建过程中除了把报表标题设为“报表”外，其他项的设置均采用默认值，命名为 Project1.rav。

报表

stu_id	name	pea_resu	fin_resu	ave_resu	class
0538010101	宋远桥	98	98	98	计网043
0538010102	张翠山	97	89	91	计网044
0538010104	周芷若	60	69	66	计网044
0538010105	赵敏	78	88	85	计网044
0538010107	俞岱岩	90	88	89	计网043

图 11.13 报表 Project1.rav

现在将使用 RvProject 组件提供的方法来动态修改该报表的三项内容：

- (1) 修改报表标题，把“报表”改为“学生成绩信息”。

(2) 增加副标题, 即在报表标题下面偏右边处加上副标题“——2005~2006 年第 1 学期”。

(3) 更改列标题, 即把英文字符的列标题改为具有概括意义的中文词组, 如把“stu_id”改为“学号”, “name”改为“姓名”等。

实现的基本思想是, 利用 RvProject 组件提供的 ProjMan.FindRaveComponent()方法分别获取 Rave 文件和 Text 文件等, 然后通过利用它们的属性修改和添加有关内容。例如, 首先使用下列语句获取报表 Report2 的 TRavePage 对象:

```
MyPage := FindRaveComponent('Report2.MainPage', nil) as TRavePage;
```

然后利用语句:

```
MyText := FindRaveComponent('Text1', MyPage) as TRaveText; //Text1 是第一列的名称
```

获取 TRavePage 对象的 Text, 进而通过 Text 的属性来修改相关项的内容。例如, 下列语句将使得第一列(stu_id 列)的列标题由“stu_id”改为“学号”, 同时把颜色设置为红色:

```
MyText.Color := clRed; //设置颜色
```

```
MyText.Text := '学号'; //设置列标题
```

FindRaveComponent()方法中运用了两个参数, 即 Report2.MainPage 和 Text1。这些参数表示什么呢? 是从何而来? 为此, 打开报表设计器(Rave Reports 6.0 BE), 并展开其右边框中的 Report Library 结点, 如图 11.14 所示。可以看出, Report2 是报表库(Report Library)中的一个报表, MainPage 则是对应着报表页, 其下包含报表标题 TitleText, 列标题: Text1、Text2、...、Text6, 以及字段名 DataText1、DataText2、...、DataText6, 可以通过这些对象名设置或访问它们的值。

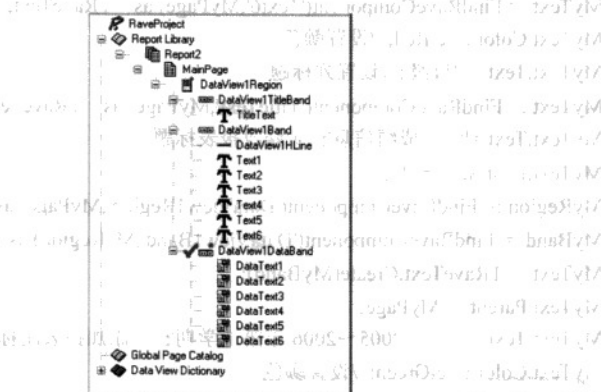


图 11.14 展开 Report Library 结点

不难看出, 上述的修改要求可以运用下面的代码来完成:

```
uses
```

```
Nevrona.Rave.RVClass, System.ComponentModel, Nevrona.Rave.RvCsRpt;
```

```
var
```

```
MyPage: TRavePage;
```

```
MyText: TRaveText;
```

```
MyBand: TRaveBand;
```

```
MyRegion: TRaveRegion;
```

Begin

```

RvProject1.Engine := RvSystem1;
RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
RvProject1.Open;
With RvProject1.ProjMan do begin
    MyPage := FindRaveComponent('Report2.MainPage',nil) as TRavePage;
    MyText := FindRaveComponent('Text1',MyPage) as TRaveText;
    MyText.Color := clRed; //设置颜色
    MyText.Text := '学号'; //设置列标题
    MyText := FindRaveComponent('Text2',MyPage) as TRaveText;
    MyText.Color := clRed; //设置颜色
    MyText.Text := '姓名'; //设置列标题
    MyText := FindRaveComponent('Text3',MyPage) as TRaveText;
    MyText.Color := clRed; //设置颜色
    MyText.Text := '平时成绩'; //设置列标题
    MyText := FindRaveComponent('Text4',MyPage) as TRaveText;
    MyText.Color := clRed; //设置颜色
    MyText.Text := '期考成绩'; //设置列标题
    MyText := FindRaveComponent('Text5',MyPage) as TRaveText;
    MyText.Color := clRed; //设置颜色
    MyText.Text := '期评成绩'; //设置列标题
    MyText := FindRaveComponent('Text6',MyPage) as TRaveText;
    MyText.Color := clRed; //设置颜色
    MyText.Text := '班级'; //设置列标题
    MyText := FindRaveComponent('TitleText',MyPage) as TRaveText;
    MyText.Text:='学生成绩信息'; //修改报表标题
    MyText.Font.Size := 24;
    MyRegion := FindRaveComponent('DataView1Region',MyPage) as TRaveRegion;
    MyBand := FindRaveComponent('DataView1Band',MyRegion) as TRaveBand;
    MyText := TRaveText.Create(MyBand);
    MyText.Parent := MyPage;
    MyText.Text:='—— 2005~2006 年第 1 学期'; //添加报表副标题
    MyText.Color := clGreen; //设置颜色
    MyText.Font.Size := 12;
    MyText.Top :=1.5;
    MyText.Left :=4.6;
    AddComponent(MyText); //增加 Text
end;
RvProject1.ExecuteReport('Report2');
RvProject1.Close;

```

End;

把上述代码编写成一个完整的程序后,运行结果如图 11.15 所示。从该图中可以看出,报表的标题等项目和属性都按照既定的方案更改过来了。

学生成绩信息

—— 2005~2006年第1学期

学号	姓名	平时成绩	期考成绩	期评成绩	班级
0538010101	宋远桥	98	98	98	计网043
0538010102	张翠山	97	89	91	计科044
0538010104	周芷若	60	69	66	计科044
0538010105	赵敏	78	88	85	计科044
0538010107	俞岱岩	90	88	89	计网043

图 11.15 报表 Project1.rav (动态修改后)

11.4.3 使用 RvNDRWriter 组件输出报表到文件中

RvNDRWriter 组件与 RvRenderHTML 组件或 RvRenderPDF 组件可分别把报表输入到 HTML 和 PDF 文件中,即以 HTML 和 PDF 文件的形式生成报表。

可以通过下列代码生成 HTML 报表文件,存放在 C:\目录下,文件名为 test.html:

```
var NDRStream: TMemoryStream;
begin
    rvproject1.Engine:=RvNDRWriter1;
    rvproject1.Open;
    rvproject1.SelectReport('report2',true);           //选择要输出的报表
    RvNDRWriter1.FileName:='tmp.ndr';
    RvNDRWriter1.StreamMode:=smfile;                 //保存成文件
    try
        NDRStream:=TMemoryStream.Create;             //建立内存流
        rvproject1.Execute;                           //输出报表数据
        NDRStream.LoadFromFile('tmp.ndr');             //将报表数据调到内存流中
        RvRenderHTML1.OutputFileName:='C:\test.html'; //存放的文件名
        RvRenderHTML1.Render(NDRStream);              //将内存流保存成 HTML 格式
        ShowMessage('生成 HTML 报表文件成功!');
    except
        ShowMessage('生成 HTML 报表文件错误!');
    end;
end;
```

而下列代码则可以生成 PDF 报表文件,存放在 C:\目录下,文件名为 test.pdf:

```
var NDRStream: TMemoryStream;
begin
    rvproject1.Engine:=RvNDRWriter1;
    rvproject1.Open;
    rvproject1.SelectReport('report2',true);
    RvNDRWriter1.FileName:='tmp.ndr';
    RvNDRWriter1.StreamMode:=smfile;
    try
```

```

NDRStream:=TMemoryStream.Create;
rvproject1.Execute;
NDRStream.LoadFromFile('tmp.ndr');
RvRenderPDF1.OutputFileName:='C:\test.pdf';
RvRenderPDF1.Render(NDRStream);
ShowMessage('生成报表文件成功!');
except
    ShowMessage('生成报表文件错误!');
end;
end;

```

需要注意的是, PDF 格式不支持中文, 所以当有中文输出的时候在 PDF 文件中将形成乱码。这是 Rave 报表对中文支持的不足之处。

11.4.4 一个动态报表设计实例

通过对前面部分的学习, 已经对 Rave 报表的制作和设计方法有了较为深入的理解。作为总结, 下面将以一个完整的实例来结束对本章的学习。

这个实例主要是对前面部分设计技术的集成, 可按照下面步骤来完成:

(1) 创建 VCL Form 应用程序, 命名为 ReportExample.dpr。在程序窗体中放入下列组件: Table、DataSource、DBGrid、RvDataSetConnection、RvProject、RvSystem、RvNDRWriter、RvRenderHTML、RvRenderPDF 组件各一个, Button 组件三个。设置各组件的属性并调节它们的大小和位置, 如表 11.3 和图 11.16 所示。

表 11.3 各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Table	Table1	DatabaseName	AccessDSN (ResultDB.mdb 的数据源)
		TableName	student
		Active	False
DataSource	DataSource1	DataSet	Table1
DBGrid	DBGrid1	DataSource	DataSource1
RvDataSetConnection	RvDataSetConnection1	DataSet	Table1
RvProject	RvProject1	对这几个组件的属性均不作任何初始化设置 (采用默认值), 而是在运行时动态设置	
RvSystem	RvSystem1		
RvNDRWriter	RvNDRWriter1		
RvRenderHTML	RvRenderHTML1		
RvRenderPDF	RvRenderPDF1		
Button	Button1	Caption	生成报表
	Button2	Caption	生成 HTML 文件
	Button3	Caption	生成 PDF 文件

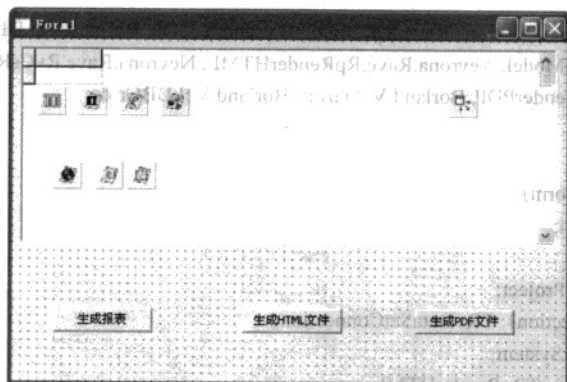


图 11.16 程序设计界面

(2) 使用 11.4.2 节创建的报表 Project1.rav, 如图 11.17 所示。

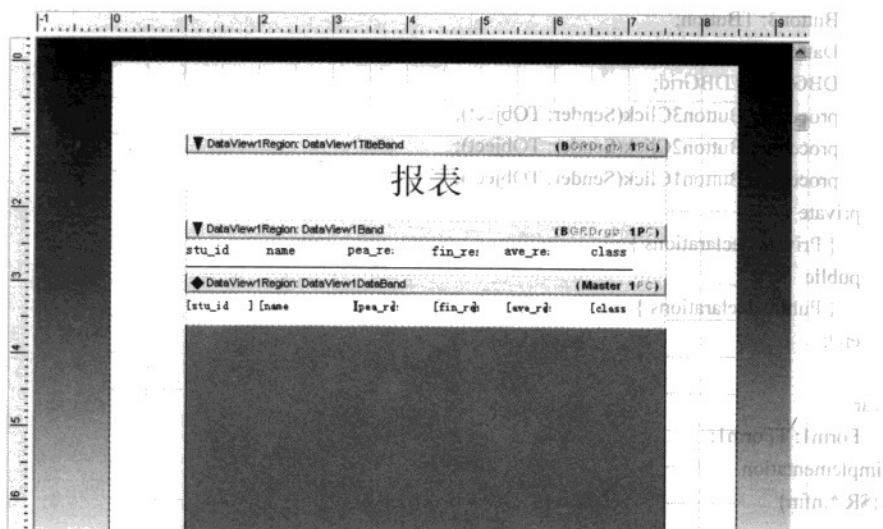


图 11.17 设计中的报表

(3) 为三个按钮编写相应的过程代码。其中,“生成报表”按钮(Button1)用于浏览数据并按照 10.4.2 小节介绍的方法动态生成并修改报表内容;“生成 HTML 文件”按钮(Button2)用于生成 HTML 报表文件,存放于 C:\目录下;“生成 PDF 文件”按钮(Button3)用于生成 PDF 报表文件,存放于 C:\目录下。

以下是应用程序 ReportExample.dpr 的单元文件 Unit1 的代码:

```
unit Unit1;
interface
```

```
uses Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, Nevrona.Rave.RpDefine, Nevrona.Rave.RpRave, StdCtrls, Nevrona.Rave.RVClass,
    Nevrona.Rave.RVProj, Nevrona.Rave.RVCsStd, Nevrona.Rave.RpBase, Nevrona.Rave.RpFiler,
    Nevrona.Rave.RpSystem, Nevrona.Rave.RpRender, Nevrona.Rave.RpRenderCanvas,
```


Nevrona.Rave.RpCon, Nevrona.Rave.RpConDS, Borland.Vcl.Db, Borland.Vcl.DBTables,
System.ComponentModel, Nevrona.Rave.RpRenderHTML, Nevrona.Rave.RvCsRpt,
Nevrona.Rave.RpRenderPDF, Borland.Vcl.Grids, Borland.Vcl.DBGrids;

type

```
TForm1 = class(TForm)
    Button1: TButton;
    Table1: TTable;
    RvProject1: TRvProject;
    RvDataSetConnection1: TRvDataSetConnection;
    RvSystem1: TRvSystem;
    RvRenderHTML1: TRvRenderHTML;
    RvNDRWriter1: TRvNDRWriter;
    RvRenderPDF1: TRvRenderPDF;
    Button2: TButton;
    Button3: TButton;
    DataSource1: TDataSource;
    DBGrid1: TDBGrid;
    procedure Button3Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;
```

var

```
Form1: TForm1;
```

implementation

```
{ $R *.nfm }
```

```
procedure TForm1.Button1Click(Sender: TObject);
```

var

```
MyPage: TRavePage;
MyText: TRaveText;
MyBand: TRaveBand;
MyRegion: TRaveRegion;
```

begin

```
Table1.Active := true;
RvProject1.Engine := RvSystem1;
RvSystem1.SystemSetups := RvSystem1.SystemSetups - [ssAllowSetup];
RvProject1.Open;
With RvProject1.ProjMan do begin
    MyPage := FindRaveComponent('Report2.MainPage', nil) as TRavePage;
    MyText := FindRaveComponent('Text1', MyPage) as TRaveText;
```

```

MyText.Color := clRed; //设置颜色
MyText.Text := '学号'; //设置列标题
MyText := FindRaveComponent('Text2',MyPage) as TRaveText;
MyText.Color := clRed; //设置颜色
MyText.Text := '姓名'; //设置列标题
MyText := FindRaveComponent('Text3',MyPage) as TRaveText;
MyText.Color := clRed; //设置颜色
MyText.Text := '平时成绩'; //设置列标题
MyText := FindRaveComponent('Text4',MyPage) as TRaveText;
MyText.Color := clRed; //设置颜色
MyText.Text := '期考成绩'; //设置列标题
MyText := FindRaveComponent('Text5',MyPage) as TRaveText;
MyText.Color := clRed; //设置颜色
MyText.Text := '期评成绩'; //设置列标题
MyText := FindRaveComponent('Text6',MyPage) as TRaveText;
MyText.Color := clRed; //设置颜色
MyText.Text := '班级'; //设置列标题
MyText := FindRaveComponent('TitleText',MyPage) as TRaveText;
MyText.Text:='学生成绩信息';
MyText.Font.Size := 24;
MyRegion := FindRaveComponent('DataView1Region',MyPage) as TRaveRegion;
MyBand := FindRaveComponent('DataView1Band',MyRegion) as TRaveBand;
MyText := TRaveText.Create(MyBand);
MyText.Parent := MyPage;
MyText.Text:='—— 2005~2006 年第 1 学期';
MyText.Color := clGreen; //设置颜色
MyText.Font.Size := 12;
MyText.Top :=1.5;
MyText.Left :=4.6;
AddComponent(MyText); //增加列标题
end;
RvProject1.ExecuteReport('Report2');
RvProject1.Close;
end;

procedure TForm1.Button2Click(Sender: TObject);
var NDRStream:TMemoryStream;
begin
    rvproject1.Engine:=RvNDRWriter1;
    rvproject1.Open;
    rvproject1.SelectReport('report2',true); //选择要打印的报表
    RvNDRWriter1.FileName:='tmp.ndr';
    RvNDRWriter1.StreamMode:=smfile; //smMemory; 保存成文件
    try
        NDRStream:=TMemoryStream.Create; //建立内存流
        rvproject1.Execute; //输出报表数据
    end;
end;

```

```

NDRStream.LoadFromFile('tmp.ndr'); //将报表数据调到内存流中
RvRenderHTML1.OutputFileName:='C:\test.html'; //存放的文件名
RvRenderHTML1.Render(NDRStream); //将内存流保存成 HTML 格式
ShowMessage('生成 HTML 报表文件成功!');
except
    ShowMessage('生成 HTML 报表文件错误!');
end;
end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var NDRStream: TMemoryStream;
begin
    rvproject1.Engine:=RvNDRWriter1;
    rvproject1.Open;
    rvproject1.SelectReport('report2',true); //选择要打印的报表
    RvNDRWriter1.FileName:='tmp.ndr';
    RvNDRWriter1.StreamMode:=smfile; //保存成文件
    try
        NDRStream:=TMemoryStream.Create; //建立内存流
        rvproject1.Execute; //输出报表数据
        NDRStream.LoadFromFile('tmp.ndr'); //将报表数据调到内存流中
        RvRenderPDF1.OutputFileName:='C:\test.pdf'; //存放的文件名
        RvRenderPDF1.Render(NDRStream); //将内存流保存成 PDF 格式
        ShowMessage('生成 PDF 报表文件成功!');
    except
        ShowMessage('生成 PDF 报表文件错误!');
    end;
end;
end;
end.

```

(4) 代码编写完成后, 编译、运行程序 ReportExample.dpr, 结果如图 11.18 所示。

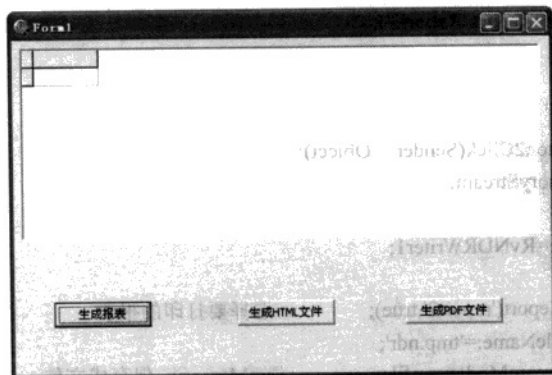


图 11.18 程序运行界面

在如图 11.18 所示的界面中, 单击“生成报表”按钮将在 DBGrid 对象中显示数据并生成如图 11.19 所示的报表; 如果单击“生成 HTML 文件”和“生成 PDF 文件”按钮将分别生成

HTML 报表文件 test1.html 和 PDF 报表文件 test1.pdf，它们都存放于 C:\ 目录中。

stu_id	name	pea_resu	fin_resu	ave_resu	class
0538010101	宋远桥	98	98	98	计网043
0538010102	张翠山	97	89	91	计科044
0538010104	周芷若	60	69	66	计科044
0538010105	赵敏	78	88	85	计科044
0538010107	俞岱岩	90	88	89	计网043

图 11.19 生成的浏览数据和报表

打开 HTML 报表文件，如图 11.20 所示。与图 11.19 相比，只是 HTML 报表没有修改报表标题和列标题，其他的均相同。但是，由于 PDF 格式不支持中文，因此打开 PDF 报表将有乱码出现（有中文的地方）。

stu_id	name	pea_resu	fin_resu	ave_resu	class
0538010101	宋远桥	98	98	98	计网043
0538010102	张翠山	97	89	91	计科044
0538010104	周芷若	60	69	66	计科044
0538010105	赵敏	78	88	85	计科044
0538010107	俞岱岩	90	88	89	计网043

图 11.20 HTML 报表文件

11.5 小结

报表设计是应用程序开发的一个重要组成部分。本章主要介绍基于 Rave 组件的报表设计和开发技术，并介绍相应的开发实例。通过对本章的学习，应掌握以下知识点：

- 熟悉常用 Rave 组件的有关属性和方法，包括 RvDataSetConnection、RvProject、RvSystem 等组件。
- 能够利用 Rave 组件进行静态和动态报表设计。
- 掌握报表设计的一般方法。

第 12 章 熟悉 Delphi 中 ASP.NET 应用程序开发

ASP.NET 是微软 .NET 技术的重要组成部分之一，它是建立在公共语言运行库上的编程框架，可用于在服务器上生成功能强大的 Web 应用程序。本章介绍了 ASP.NET 的新特点、ASP.NET 开发技术，以及基于 ASP.NET 技术的应用程序开发技能。本章内容涉及如下要点：

- ASP.NET 结构、运行环境的搭建和配置。
- ASP.NET 工程的文件结构。
- 常用 ASP.NET 组件的属性和方法。
- Web Form 的基本指令、语法及其 ASP.NET 内置对象。
- 执行页导航及参数传递方法。
- 基于 BDP 的 ASP.NET 数据库开发技术。

12.1 了解 ASP.NET

12.1.1 ASP.NET 的特点

ASP.NET 是 ASP 的升级版，但更重要的是它对 ASP 进行了许多根本性的改进，吸收了 ASP 版本的优点并参照 Java、VB 语言的开发优势加入了许多新的特色。ASP.NET 是微软发展的新型体系结构 .NET 的一部分，是 .NET 构架中最重要的技术之一。它是建立在通用语言运行库（Common Language Runtime）之上的程序构架，通常在 Web 服务器后端为用户提供强大的企业级 Web 应用服务，它在语法上大致与 ASP 兼容，还另外提供新的程序设计模型和基础结构，以开发延展性更大、稳定性更佳、安全性更强的应用程序。

ASP.NET 具有如下一些特点：

(1) 高效的执行效率。与 ASP 不同，ASP.NET 不再通过解释来执行，而是通过与 NGWS（MicroSoft's Next Generation Web Services）兼容的编译器编译来执行，即将程序在服务器端首次运行时进行编译，这样的执行效率就比边解释边执行的效率高得多。

(2) 具有强大的功能和适应性。基于通用语言编译运行的 ASP.NET 程序，可以在 Web 应用软件开发者的几乎所有平台上运行。它支持多种语言编程，如 C#、VB、Jscript 等，可以开发出功能强大的应用程序。另外，ASP.NET 程序是利用 ADO.NET 来访问数据库。ADO.NET 引入了数据集（DataSet），这是它的创新点。一个数据集是内存中提供数据关系图的高速缓冲区，由数据提供者填充或回填到数据库中。但数据集对数据源是独立的，所以不同的程序模块、不同的语言代码都可以通过操作数据集来实现对多种数据库的访问。因此，ASP.NET 程序可以在各种平台上实现对各种不同数据库的访问，具有很强的数据访问功能。

(3) 具有良好的可扩展性。在进行 ASP.NET 程序设计时，程序员可以在代码中自己定义

“plug-in”的模块，加入自己开发的组件，从而提高自己工作的可重用性。

(4) 多种支持语言。ASP 程序通常要用脚本语言编写，仅限于 VScript 和 Jscript。但是对于 ADO.NET 程序，还可以用 C#（C++和 Java 的结合）、VB 或者 Object Pascal 来编写。

(5) 提供丰富的服务器控件。微软已经编写了大量的 .NET 组件，在 ASP.NET 程序中可以直接引用，如列表框、日历控件等。

12.1.2 ASP.NET 结构

ADO.NET 体系结构如图 12.1 所示，它主要是由 HTML 控制和服务控制的 Web 窗体页、ADO.NET 服务器控制、Web 服务、后台代码逻辑及编译好的 DLL 文件组成。

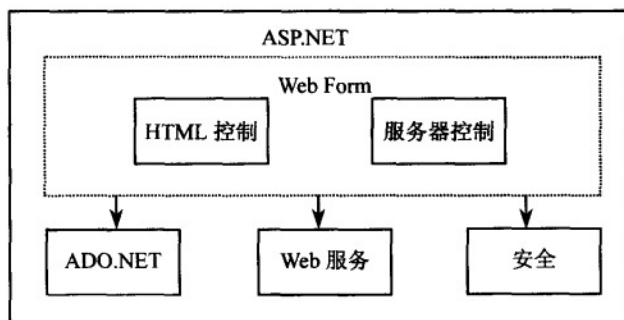


图 12.1 ADO.NET 体系结构

ASP.NET 应用程序与 Web 页（Web 客户端）之间的联系要通过 Web 服务器 IIS（Internet Information Services）来完成。IIS 是 Microsoft 随操作系统（如 Windows NT/2000、Windows XP 等）提供的一种组件，但默认情况下并不安装。ASP.NET 应用程序、客户端程序、IIS、.NET 框架及 Windows 操作系统之间的关系如图 12.2 所示。

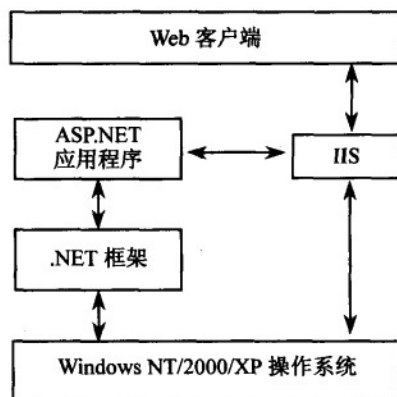


图 12.2 ASP.NET 应用程序与 IIS 等之间的关系

ASP.NET 应用程序是建立在 .NET 框架技术基础上的，因此 ASP.NET 应用程序可以充分利用由 .NET 框架提供的各种组件和服务，以减少编程工作量和降低开发成本。

12.2 配置运行环境及创建 ASP.NET 应用程序

12.2.1 配置运行环境

1. ASP.NET 的开发和运行环境

(1) ASP.NET 运行环境。在下列环境下可以运行客户端和服务器应用程序：ASP.NET Windows 2000 (Professional、Server 和 Advanced Server)、Windows XP Professional (商用版)、Windows Server 2003 系列。

(2) ASP.NET 开发环境。开发 ASP.NET 服务器应用程序需要下列软件：

- Windows 2000 Server 或 Advanced Server (并安装 Service Pack 2)、Windows XP Professional (商用版) 或 64 位版, 或 Windows Server 2003 系列产品的一项。
- MDAC 2.7 for Data。
- Internet Information Services (IIS)。

2. 硬件配置

- CPU: Intel Pentium II-class 300 MHz (最好 Intel Pentium III-class 600 MHz)。
- 内存: 128MB 及以上。
- 磁盘空间: 250 MB (完全安装) 和 155 MB (快速安装)。
- 显示: 一般为 800×600, 最好是 1024×800, 256 色。


12.2.2 安装 IIS 5.1


IIS (Internet Information Services) 是开发 ASP.NET 应用程序不可或缺的 Windows 系统组件, 在程序运行时充当 Web 服务器的角色。

在安装 Windows 系统时, 默认是不安装 IIS 的。这需要在安装 Windows 系统时手动选择 IIS 组件, 或者在系统安装完毕后再安装 IIS 组件。下面将介绍在系统安装完毕后再安装 IIS 组件的方法。

可以按照下面的步骤完成对 IIS 5.1 的安装。

(1) 在 Windows 系统中选择“开始”→“控制面板”命令, 打开“控制面板”对话框; 在此对话框中点击超链接“添加/删除程序”, 打开“添加或删除程序”对话框, 然后单击该对话框中左边的“添加/删除 Windows 组件”图标, 在系统扫描组件的安装情况后弹出“Windows 组件向导”对话框。

在“Windows 组件向导”对话框中, 如果复选框 ☐  Internet 信息服务 (IIS) 没有被选中, 这表明系统中还没有安装 IIS 组件。

(2) 选中复选框 ☐  Internet 信息服务 (IIS), 单击“下一步”按钮, IIS 将进入文件复制的安装过程。在这个过程中, 安装程序将提示插入 Windows 系统安装盘, 只要把 Windows 系统安装盘插入光驱即可。

(3) 安装文件复制完成以后, 将出现安装完成提示框, 单击“完成”按钮, IIS 的整个安装过程结束。

上述安装步骤完成以后, 打开 IE 浏览器, 在其地址栏中输入“http://localhost”, 如果弹出

如图 12.3 所示的两个页面, 则说明 IIS 已经成功安装。

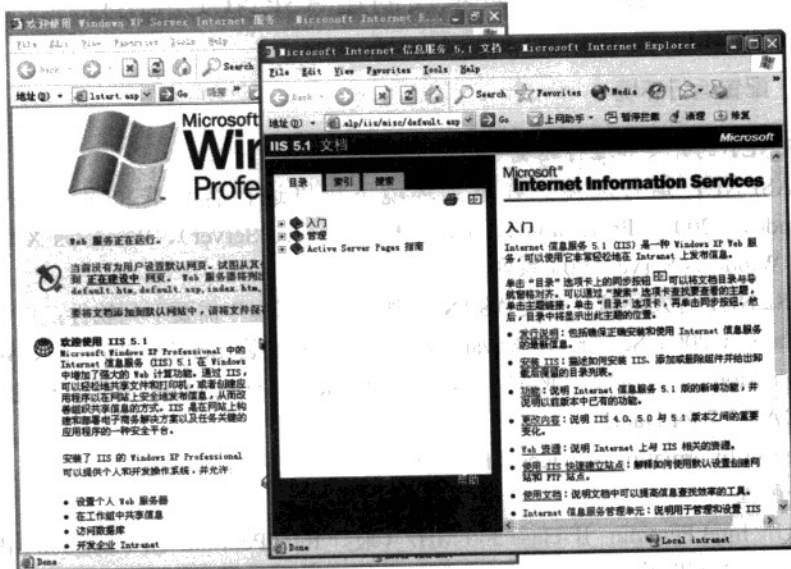


图 12.3 IIS 的默认 Web 站点的首页

成功安装 IIS 后, 将产生如下的目录: C:\inetpub\wwwroot\ (假设 C: 为系统盘), 它就是 IIS 的默认 Web 发布目录, 所有发布的文件都放在这个目录下。

12.2.3 创建 ASP.NET 应用程序

在安装了 IIS 以后, 就可以利用 Delphi 2005 开发 ASP.NET 应用程序了。在这里, 首先介绍创建 ASP.NET 应用程序的基本方法, 为后面的学习奠定基础。

(1) 在 Delphi 2005 IDE 中, 选择 File→New→ASP.NET Web Application - Delphi for .NET 命令, 然后打开如图 12.4 所示的对话框。

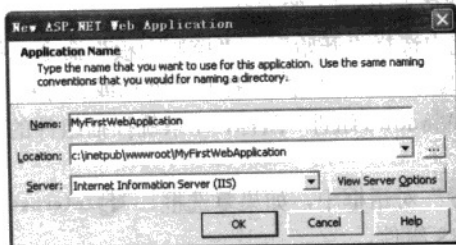


图 12.4 创建 ASP.NET 应用程序

在此对话框中, 有下面三项需要设置。

- 应用程序名: 在 Name 文本框中设置 ASP.NET 应用程序名。
- 应用程序保存位置: 在 Location 下拉列表框中选择应用程序要保存的位置, 一般只能在 IIS 默认的目录 (C:\inetpub\wwwroot\) 及其子目录下; 如果保存在其他目录中,

则要设置虚拟目录。

- Web 服务器: 在 Server 下拉列表框中选择应用程序的 Web 服务器, ASP.NET 应用程序只能选择 IIS。

(2) 设置结果如图 12.4 所示, 单击 OK 按钮, 名为 MyFirstWebApplication 的 ASP.NET 应用程序被创建。然后, 打开工具面板中的 WebControls 标签, 从中选中 Button 组件并把它拖放到设计页面中, 将其 Text 属性值改为“你好!”; 此外还把 Label 组件拖到设计页面中, 把它的 Font.Size 属性值设置为 Medium, 并适当调整各组件的位置和大小, 结果如图 12.5 所示。

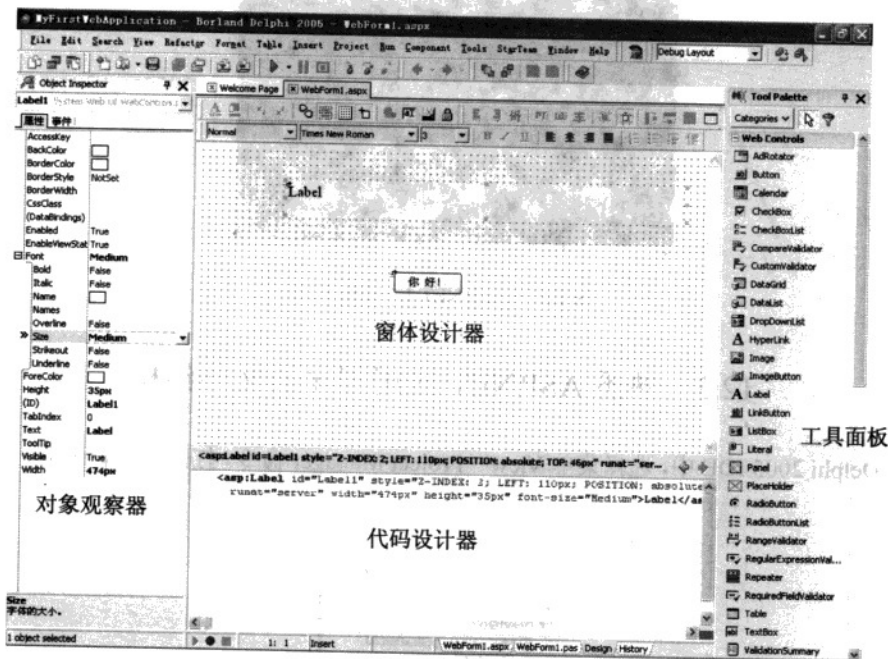


图 12.5 ASP.NET 应用程序设计界面

(3) 在设计界面中双击“你好!”按钮, 在它的 Click 方法中添加如下代码:
procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
begin

Label1.Text := Button1.Text + ', 这是我的第一个 ASP.NET 应用程序!';
end;

(4) 保存该 ASP.NET 应用程序 MyFirstWebApplication, 然后单击运行按钮 运行该程序, 并在运行界面中单击“你好!”按钮, 结果在 IE 浏览器中被打开。

在第一次运行 ASP.NET 应用程序时可能出现如图 12.6 所示的调试错误提示(除非是 IIS6.0)。

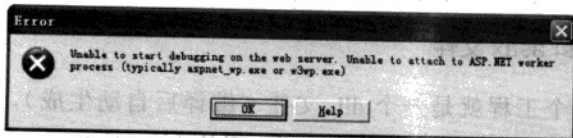


图 12.6 调试错误提示

一个解决的办法就是选择 Windows 系统菜单“开始”→“运行”命令，打开运行对话框，运行下列命令：

```
C:\windows\microsoft.net\framework\v1.1.4322\aspnet_regiis.exe -i
```

然后会打开如图 12.7 所示的 ASP.NET 1.1 安装界面。等到该界面自动消失后，ASP.NET 1.1 安装成功，这时就可以运行 ASP.NET 应用程序了。

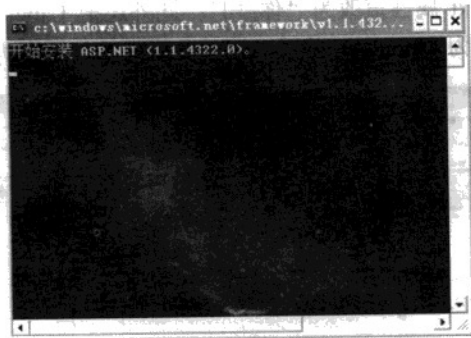


图 12.7 ASP.NET 1.1 安装界面

12.3 熟悉 ASP.NET 应用程序的文件结构

在 Delphi 2005 IDE 中，选择菜单 View→Project Manager 命令，打开工程管理器，如图 12.8 所示。



图 12.8 工程管理器

从图 12.8 中可以看出，一个 ASP.NET 应用程序由.dll 文件、.aspx 文件、.pas 文件、Web.config 文件及.resx 文件等部分组成，并且它们有分明的层次结构。

12.3.1 .dll 文件和.aspx 文件

从整体上看，整个工程就是一个.dll 文件（编译后自动生成），进一步又可以分为 References、Deployment、Global.asax、Web.config 和 WebForm1.aspx。其中，References 部分

包含了工程所引用的.dll 文件, 在此可以添加或删除.dll 文件。

Global.asax 文件又称为 ASP.NET 应用程序文件, 在运行时分析 Global.asax 文件并将其编译到一个动态生成的 .NET Framework 类, 该类是从 `HttpApplication` 基类派生出来的。更改活动的 Global.asax 文件时 (保存后), ASP.NET 页框架会检测到该文件已被更改。它完成应用程序的所有当前请求, 将 `Application_OnEnd` 事件发送到任何侦听器, 并重新启动应用程序域。实际上, 这会重新启动应用程序, 关闭所有浏览器会话并刷新所有状态信息。当来自浏览器的下一个传入请求到达时, ASP.NET 页框架将重新分析并重新编译 Global.asax 文件, 并引发 `Application_OnStart` 事件。

Global.asax 文件拒绝对它的任何直接 URL 请求, 因此外部用户无法下载或查看在该文件中编写的代码。Global.asax 文件驻留在基于 ASP.NET 的应用程序的根目录中, 是可选文件。如果一个 ASP.NET 应用程序不定义该文件, 那么程序默认没有定义任何应用程序或会话事件处理程序, 即不对这些事件作任何响应。

Global.pas 文件是 Global.asax 文件的实现版本, 其代码如下 (对每一个事件的引发机制作了说明):

```
unit Global;
interface

uses
  System.Collections, System.ComponentModel,
  System.Web, System.Web.SessionState;
type
  TGlobal = class(System.Web.HttpApplication)
  {$REGION 'Designer Managed Code'}
  strict private
    procedure InitializeComponent;
  {$ENDREGION}
  strict protected
    procedure Application_Start(sender: System.Object; e: EventArgs);
    procedure Session_Start(sender: System.Object; e: EventArgs);
    procedure Application_BeginRequest(sender: System.Object; e: EventArgs);
    procedure Application_EndRequest(sender: System.Object; e: EventArgs);
    procedure Application_AuthenticateRequest(sender: System.Object; e: EventArgs);
    procedure Application_Error(sender: System.Object; e: EventArgs);
    procedure Session_End(sender: System.Object; e: EventArgs);
    procedure Application_End(sender: System.Object; e: EventArgs);
  private
    { Private Declarations }
  public
    constructor Create;
  end;

implementation
{$REGION 'Designer Managed Code'}
/// <summary>
```

```
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TGlobal.InitializeComponent;
begin
end;
{$SENDREGION}
constructor TGlobal.Create;
begin
    inherited;
    // Required for Designer support
    InitializeComponent;
    // TODO: Add any constructor code after InitializeComponent call
end;

procedure TGlobal.Application_Start(sender: System.Object; e: EventArgs);
begin
    // ASP.NET 程序重新启动或重新编译 Global.asax 文件时引发该事件
end;

procedure TGlobal.Session_Start(sender: System.Object; e: EventArgs);
begin
    //会话重新启动时引发该事件
end;

procedure TGlobal.Application_BeginRequest(sender: System.Object; e: EventArgs);
begin
    //任意一个请求开始时都会引发该事件
end;

procedure TGlobal.Application_EndRequest(sender: System.Object; e: EventArgs);
begin
    //任意一个请求结束时都会引发该事件
end;

procedure TGlobal.Application_AuthenticateRequest(sender: System.Object; e: EventArgs);
begin
    //试图进行身份验证时引发该事件
end;

procedure TGlobal.Application_Error(sender: System.Object; e: EventArgs);
begin
    //在应用程序出现运行错误时引发该事件
end;

procedure TGlobal.Session_End(sender: System.Object; e: EventArgs);
begin
```

```
//会话结束时引发该事件
```

```
end;
```

```
procedure TGlobal.Application_End(sender: System.Object; e: EventArgs);
```

```
begin
```

```
//应用程序结束时引发该事件
```

```
end;
```

```
end.
```

12.3.2 Web.config 文件

Web.config 文件是一个 XML 文本文件,它是 ASP.NET 应用程序的配置文件。每创建一个 ASP.NET 应用程序都会在根目录下默认自动产生一个 Web.config 文件。例如,在上节创建了 ASP.NET 应用程序 MyFirstWebApplication 后,产生的 Web.config 的代码如下:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation
      defaultLanguage="c#"
      debug="true">
      <assemblies>
      </assemblies>
    </compilation>
    <httpModules>
    </httpModules>
    <customErrors
      mode="RemoteOnly"
    />
    <authentication mode="Windows" />
    <trace
      enabled="false"
      requestLimit="10"
      pageOutput="false"
      traceMode="SortByTime"
      localOnly="true"
    />
    <sessionState
      mode="InProc"
      stateConnectionString="tcpip=127.0.0.1:42424"
      sqlConnectionString="data source=127.0.0.1;user id=sa;password="
      cookieless="false"
      timeout="20"
    />
    <globalization
      requestEncoding="utf-8"
      responseEncoding="utf-8"
```

```
</>  
</system.web>  
</configuration>
```

上面的 Web.config 文件具有一定的代表性，下面来分析它的结构。

1. 在 Web.config 文件中所有的配置信息

运行都包含在下列标记之间：

```
<configuration>  
<system.web>  
和  
</system.web>  
</configuration>
```

2. <authentication>标记

该标记用于配置 ASP.NET 身份验证支持，有四种方式，即 Windows、Forms、PassPort、None。例如，上例中该标记设置如下：

```
<authentication>  
<authentication mode="Windows">  
</authentication>
```

表示使用 Windows 系统登录验证方式验证。注意，<authentication>和</authentication>必须成对出现，否则要使用以下形式：

```
<authentication mode="Windows" />
```

如果 mode 的值为“Form”，则表示使用基于窗体（Forms）的身份验证配置，例如，下列的<authentication> 标记的作用是：当没有登录的用户访问需要身份验证的网页，网页自动跳转到登录网页 index.aspx。

<authentication> 标记设置为以下形式：

```
<authentication mode="Forms" >  
    <forms loginUrl="index.aspx" name=".FACookie"/>  
</authentication>
```

其中，元素 loginUrl 用于设置登录网页的名称，name 表示 Cookie 名称。

3. <compilation>标记

该标记用于创建 ASP.NET 程序使用的所有编译设置。例如，下列标记表示要把调试符号插入到编译页中：

```
<compilation  
    defaultLanguage="c#"  
    debug="true">  
</compilation>
```

默认的 debug 属性值为“true”。显然，由于把调试符号插入到编译页，所以生成的可执行文件可能很大，导致执行速度慢，所以在程序编译完成交付使用之后应将 debug 属性值设为 false。

4. <customErrors>标记

该标记用于设置自定义信息。当 customErrors mode = "On"或"RemoteOnly"时，则表示启用用户自定义错误信息；如果设置为"Off"时则表示禁用自定义错误信息。但该标记不适用于 XML Web services 中发生的错误。

例如, 下面的标记设置可以使得在发生错误时, 将网页跳转到自定义的错误页面 `ErrorPage.aspx`:

```
<customErrors defaultRedirect="ErrorPage.aspx" mode="RemoteOnly" />
```

其中, 元素 `defaultRedirect` 用于指定自定义错误网页的名称; “`mode="RemoteOnly"`”表示对不在本地 Web 服务器上运行的用户显示自定义 (友好的) 信息。出于安全考虑, 建议使用该设置。

5. <httpRuntime>标记

该标记用于配置 ASP.NET HTTP 运行库设置。例如, 规定用户上传文件最大为 10M, 最长时间为 500 秒, 最多请求数为 10。

```
<httpRuntime maxRequestLength="10240" executionTimeout="500" appRequestQueueLimit="10"/>
```

该标记可以在站点、应用程序和子目录级别设置。

6. <pages>标记

该标记用于设置是否启用会话状态、视图状态, 是否检测用户的输入等, 其作用范围仅限于当前 Web 页。例如, 下列标记表示要检测用户在浏览器输入的内容中是否存在潜在的危险数据, 以及检查加密的视图状态:

```
<pages buffer="true" enableViewStateMac="true" validateRequest="true"/>
```

7. <sessionState>标记

该标记用于配置会话状态设置, 如是否启用会话状态、会话状态保存位置等。例如:

```
<sessionState
    mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user id=sa;password="
    cookieless="false"
    timeout="20"
/>
```

其中, `mode="InProc"`表示要在本地储存会话状态; `cookieless="false"`表示如果用户浏览器不支持 cookie 时将会启用会话状态; `timeout="20"`则表示, 会话可以处于空闲状态的最长时间为 20 分钟。

8. <globalization>标记

此标记用于配置 ASP.NET 应用程序的全球化设置。例如:

```
<globalization
    requestEncoding="utf-8"
    responseEncoding="utf-8"
/>
```

9. <trace>标记

该标记用于配置 ASP.NET 跟踪服务。例如, 上一节创建的 `MyFirstWebApplication` 应用程序, 其对应的 `<trace>` 标记设置如下:

```
<trace
    enabled="false"
    requestLimit="10"
    pageOutput="false"
    traceMode="SortByTime"
```



```
localOnly="true"
```

```
</>
```

其中, `enabled="false"` 表示不启用跟踪服务; `requestLimit="10"` 表示在服务器上存储的跟踪请求的数目; `pageOutput="false"` 表示只能通过跟踪的实用工具才能访问跟踪的输出; `traceMode="SortByTime"` 表示按照处理跟踪的顺序来显示跟踪信息; `localOnly="true"` 表示跟踪查看器 (trace.axd) 只用于宿主 Web 服务器。

12.3.3 Web 窗体文件

Web 窗体文件由两种不同类型的文件组成, 即 .aspx 文件和 pas 文件, 前者是 Web 页面文件, 后者是代码文件 (又称代码隐藏类文件)。Web 页面文件用于创建可视元素, 如静态 HTML 和/或 ASP.NET 服务器控件; 代码文件则视 Web 页面文件的编程逻辑, 在该文件中编写的逻辑可以使用 Object Pascal 来编写。实际上, 这两个文件是一个有机的整体: 在窗体页中创建可视元素会自动在代码文件中生成相应的代码, 设计代码文件是对窗体页功能的扩展。

对于上一节创建的 MyFirstWebApplication 应用程序, 其 Web 窗体文件包含 WebForm1.aspx 和 WebForm1.pas。其中, WebForm1.aspx 的代码如下:

```
<form runat="server">
  <asp:Button id="Button1"
    style="Z-INDEX: 1; LEFT: 238px; POSITION: absolute; TOP: 150px"
    runat="server" height="27px" width="83px" text="你 好! "></asp:Button>
  <asp:Label id="Label1" style="Z-INDEX: 2; LEFT: 110px; POSITION: absolute; TOP: 46px"
    runat="server" height="35px" width="474px" font-size="Medium">Label</asp:Label>
</form>
```

.aspx 文件有两种设计模式, 一种是可视化设计模式, 另一种是代码设计模式。前者是通过拖放组件的方法来设计应用程序的窗体, 后者则是通过编写代码来完成代码的窗体设计。

WebForm1.pas 的代码如下:

```
unit WebForm1;
interface

uses
  System.Collections, System.ComponentModel,
  System.Data, System.Drawing, System.Web, System.Web.SessionState,
  System.Web.UI, System.Web.UI.WebControls, System.Web.UI.HtmlControls;

type
  TWebForm1 = class(System.Web.UI.Page)
  {$REGION 'Designer Managed Code'}
  strict private
    procedure InitializeComponent;
    procedure Button1_Click(sender: System.Object; e: System.EventArgs);
  {$ENDREGION}
  strict private
    procedure Page_Load(sender: System.Object; e: System.EventArgs);
  strict protected
```

```
    Button1: System.Web.UI.WebControls.Button;
    Label1: System.Web.UI.WebControls.Label;
    procedure OnInit(e: EventArgs); override;
private
    { Private Declarations }
public
    { Public Declarations }
end;
implementation
{$REGION 'Designer Managed Code'}

/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWebForm1.InitializeComponent;
begin
    Include(Self.Button1.Click, Self.Button1_Click);
    Include(Self.Load, Self.Page_Load);
end;
{$ENDREGION}

procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
begin
    // TODO: Put user code to initialize the page here
end;

procedure TWebForm1.OnInit(e: EventArgs);
begin
    //
    // Required for Designer support
    //
    InitializeComponent;
    inherited OnInit(e);
end;

procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
begin
    Label1.Text := Button1.Text + ', 这是我的第一个 ASP.NET 应用程序! ';
end;
end.
```

从上面的代码可以看出，ASP.NET 应用程序的功能主要是由 WebForm1.pas 文件的代码来实现的，也就是说，WebForm1.pas 是程序员的“用武之地”。显然，这一编程语言就是 Object Pascal。所以，从这里可以看出，Delphi 2005 是一种提供运用 Object Pascal 来开发 ASP.NET 应用程序的平台。

在编译时，项目中所有的代码隐藏类文件都被编译成项目动态链接库(.dll)文件。.aspx 文件也会被编译，但编译方式稍有不同。当用户第一次浏览到 .aspx 页时，ASP.NET 自动生成表示该页的.NET 类文件，并将其编译成另一个.dll 文件。为.aspx 页生成的类从被编译成项目.dll 文件的代码隐藏类继承。当用户请求 Web 页 URL 时，.dll 文件将在服务器上运行并动态地生成 HTML 输出。

12.4 熟悉常用的 ASP.NET 组件










从前面创建的 ASP.NET 应用程序来看，其开发过程也是基于组件的开发和设计过程。所以，熟悉常用 ASP.NET 组件的有关属性和方法是开发 ASP.NET 应用程序的基本要求。本节将主要介绍一些常用的 ASP.NET 组件。

12.4.1 Button 和 ImageButton 组件

与 VCL 组件一样，Button 也是 Web 窗体页中最常用的按钮组件之一。但在 Web 窗体页中一般要求组件具有更好、更漂亮的外观，所以 Button 组件提供了丰富的用于外观设置的属性。

- (1) BackColor 属性。该属性用于设置 Button 组件的背景颜色。
- (2) BorderColor 属性。该属性用于设置 Button 组件的边框颜色。
- (3) BorderWidth 属性。BorderWidth 属性用于设置 Button 组件边框的宽度，单位为像素。
- (4) BorderStyle 属性。该属性用于设置 Button 组件边框的类型，当 BorderWidth 属性值被设置为大于或者等于 1 时该属性才可见。它可取十个值，各值产生的边框效果如表 12.1 所示。

表 12.1 BorderStyle 属性设置的效果 (BorderWidth 属性值为 4)

BorderStyle 属性值	组件边框效果
NotSet	
None	
Dotted	
Dashed	
Solid	
Double	
Groove	
Ridge	
Inset	
Outset	

如果要使组件返回系统默认的形状，须把 BorderWidth 属性值设置为空（不是 0）。

- (5) CausesValidation 属性。CausesValidation 属性值是布尔类型，其默认值为 true，表示

单击 Button 组件时要执行页面验证, 常用于提交功能的按钮; 如果 CausesValidation 属性值被设置为 false, 那么单击 Button 组件时不会执行页面验证, 常用于重置或取消功能的按钮。

(6) CommandName 属性。当多个 Button 组件联合使用时, 就会用到 CommandName 属性值。该属性值可以是任意一个有效的标识符 (字符串型)。其通常的用法是: 对多个联合使用的 Button 组件, 为每个设置一个 CommandName 属性值 (彼此不同), 然后使它们共用一个 Command 方法; 在 Command 方法中通过判断 Button 组件的 CommandName 属性值来获知当前是哪一个 Button 组件被单击, 从而进行相应的操作。通过下面的例子可以理解这一点。

创建一个 ASP.NET 应用程序。在设计窗体中放入两个 Button 组件和一个 Label 组件, 然后把 Button 组件的 CommandName 属性值分别设置为 Command_Button1 和 Command_Button2, 最后把两个 Button 组件的 Command 方法名都设为 ButtonCommand (使得它们共享一个方法), 并双击其中一个则进入为事件添加代码的地方, 添加的代码如下:

```
procedure TWebForm1.ButtonCommand(sender: System.Object; e:
    System.Web.UI.WebControls.CommandEventArgs);
begin
    if e.CommandName = 'Command_Button1' then Label1.Text := '您单击了按钮 1!';
    if e.CommandName = 'Command_Button2' then Label1.Text := '您单击了按钮 2!';
end;
```

运行该 ASP.NET 应用程序, 不断地单击这两个按钮, Label 组件将轮流显示“您单击了按钮 1”和“您单击了按钮 2”, 图 12.9 显示了其中的一个效果。

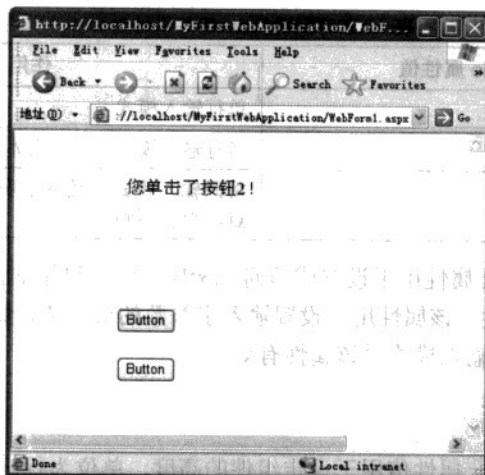


图 12.9 CommandName 属性的应用

实际上, 在窗体设计页中对组件属性的设置最终都反映到页面的源文件中, 以下是上述设置的结果:

```
<form runat="server">
    <asp:Label id="Label1" style="Z-INDEX: 2; LEFT: 110px; POSITION: absolute; TOP: 46px"
        runat="server" height="35px" width="474px" font-size="Medium"
        borderwidth="0px">Label</asp:Label>
    <asp:Button id="Button1"
```

```

style="Z-INDEX: 8; LEFT: 102px; POSITION: absolute; TOP: 174px"
runat="server" text="Button" commandname="Command_Button1"></asp:Button>
<asp:Button id="Button2"
style="Z-INDEX: 9; LEFT: 102px; POSITION: absolute; TOP: 222px"
runat="server" text="Button" commandname="Command_Button2"></asp:Button>
<asp:TextBox id="TextBox1"
style="Z-INDEX: 10; LEFT: 286px; POSITION: absolute; TOP: 118px"
runat="server" borderstyle="None"></asp:TextBox>
</form>

```

不难看出在源文件中各组件属性的设置情况。当然，完全可以在此修改组件的属性值，但对初学者来说一般是在窗体设计页中进行设计。

ImageButton 组件与 Button 组件的属性基本一样，不同的是 ImageButton 组件增加了 ImageUrl 属性、ImageAlign 属性和 AlternateText 属性。这三个属性分别用于在 ImageButton 组件上添加图片、设置图片的对齐方式和设置组件的文本字符。

12.4.2 TextBox 组件

TextBox 组件的属性很多都是与 Button 组件的相同，在此主要介绍其常用的主要属性。

(1) TextMode 属性。TextMode 属性用于设置文本框的输入模式，它有三种值，其说明如表 12.2 所示。

表 12.2 TextMode 属性值的含义

TextMode 属性值	作用
SingleLine	单行输入模式
MultiLine	多行输入模式，即在文本框中可以输入多行
PassWord	密码输入模式（输入字符时只看到一个黑圆点，而看不到字符）


(2) Text 属性。Text 属性用于设置或访问 TextBox 组件显示的内容。

(3) MaxLength 属性。该属性用于设置输入字符数的最大值，默认值为 0，表示不限制。当 TextBox 组件设为单行输入模式时该属性有效。

12.4.3 ListBox 组件

(1) Row 属性。Row 属性值用于设定组件的高度，单位为行。例如，如果 Row 属性值为 3，则 ListBox 组件的高度为 3 行。

(2) SelectionMode 属性。SelectionMode 属性值有两种：Simple 和 Multiple。默认值为 Simple，表示在程序运行时只能选择一行，如果为 Multiple 时则可选择多行。

(3) Items 属性。Items 属性用于为 ListBox 组件设置初值，方法是单击该属性项右边的省略号按钮，打开如图 12.10 所示的“ListItem 集合编辑器”对话框。在此，可以通过单击“添加”按钮或者“移除”按钮分别为 ListBox 组件添加项目和删除项目。

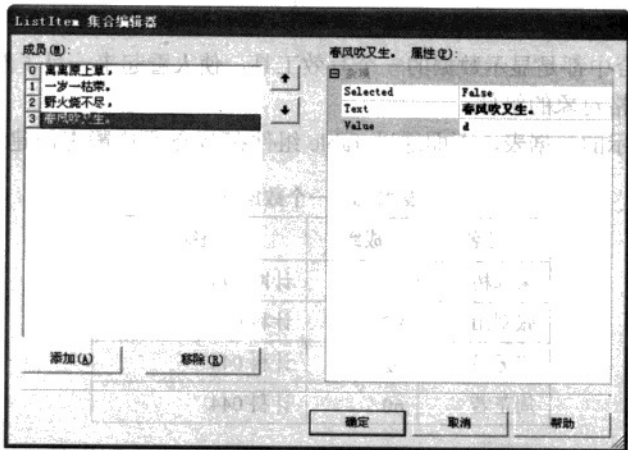


图 12.10 “ListItem 集合编辑器”对话框

12.4.4 CheckBox 和 CheckBoxList 组件

CheckBox 组件用于在 Web 面上创建复选框，其主要属性是 Checked。当复选框被选中时，Checked 属性值为 true；当复选框没有被选中时，Checked 属性值为 false。在编程时，可以通过设置或访问 Checked 属性值来实现相应的功能。当选中复选框（Checked 属性值由 true 变为 false，或者由 false 变为 true）时，CheckedChanged 事件会被触发。

CheckBoxList 组件与 CheckBox 组件基本类似，不同的是，CheckBoxList 组件可由 Items 属性嵌入多个复选框，它没有 Checked 属性。当选中其中一个复选框时，被触发的是 SelectedIndexChanged 事件，而没有 CheckedChanged 事件。

下面这一段代码则是把 CheckBoxList1 中被选中的复选框对应的 Text 属性值添加到 ListBox1。从中不难看出 CheckBoxList 组件的使用方法。

```
var s:string;  
    i,count:integer;  
begin  
    count := CheckBoxList1.Items.Count;  
    ListBox1.Items.Clear;  
    for i:=0 to count-1 do  
    begin  
        if CheckBoxList1.Items.Item[i].Selected then  
        begin  
            s := CheckBoxList1.Items.Item[i].ToString;  
            ListBox1.Items.Add(s);  
        end;  
    end;  
end;
```

12.4.5 RadioButton 组件、RadioButtonList 组件和 Table 组件

RadioButton 和 RadioButtonList 组件分别与 CheckBox 和 CheckBoxList 组件的使用方法类似。不同的是，在一个容器中只能选中一个 RadioButton，在一个 RadioButtonList 组件中也只

能选中其中的一个单选框。

表格在许多场合中都是显示数据的一个有效工具,使人看起来一目了然。Table 正是用于在 Web 页中创建表格对象的组件。

对于表 12.3 所示的数据表,下面通过 Table 组件在 Web 页中静态创建这个表。

表 12.3 一个数据表

姓名	成绩	班级
宋远桥	98	计网 043
张翠山	97	计科 044
张无忌	92	计科 044
周芷若	60	计科 044

首先在窗体设计页中加入 Table 组件,并适当调整它的大小和位置,如图 12.11 所示。

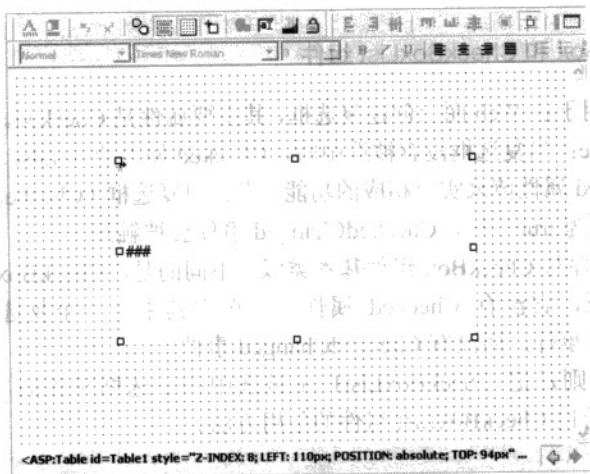




图 12.11 加入 Table 组件

然后选中 Table 组件,在对象观察器中找到 Rows 属性项并单击其右边的省略号按钮 , 打开 TableRow 集合编辑器。

在 TableRow 集合编辑器中单击“添加”按钮,然后在右边出现的 TableRow 属性框中找到 Cells 属性项并单击其右边的省略号按钮 , 将打开“TableCell 集合编辑器”对话框。在该对话框中单击“添加”按钮,然后在右边的“TableCell 属性”框中找到 Text 属性,并设置其值为“姓名”,同时还把 Font.Bold 属性值设为 true (第一行用黑体字),这样就建立值为“姓名”(黑体)的表单元。

接着继续在“TableCell 集合编辑器”对话框中单击“添加”按钮,然后在右边的“TableCell 属性”框中找到 Text 属性,并设置其值为“成绩”,同时还把 Font.Bold 属性值设为 true,这样又建立值为“成绩”(黑体)的表单元;用同样方法建立值为“班级”(黑体)的表单元,如图 12.12 所示。

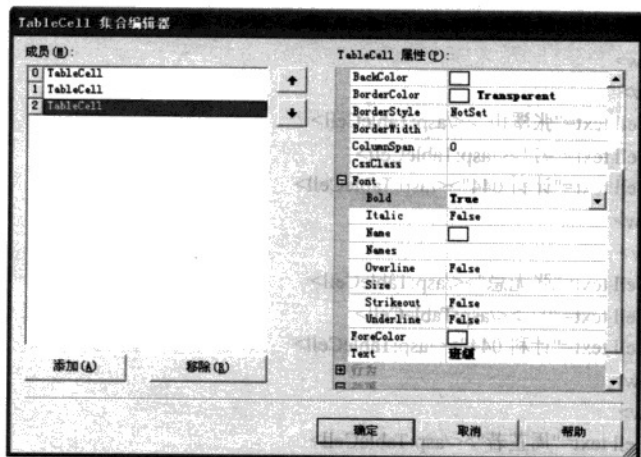


图 12.12 “TableCell 集合编辑器”对话框（创建了三个表单元）

在如图 12.12 所示的“TableCell 集合编辑器”对话框中单击“确定”按钮，就创建了表 12.3 的第一行。

用上述类似的方法依次创建表 12.3 中的其他数据记录，不同的是不要把各个表单元的字

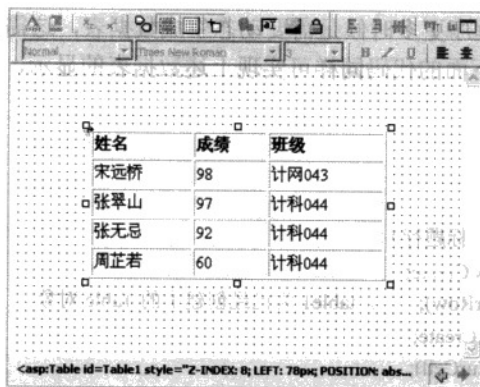


图 12.13 设计结果

该 Web 页对应的代码如下：

```
<asp:Table id="Table1" style="Z-INDEX: 8; LEFT: 78px; POSITION: absolute; TOP: 62px"
runat="server" width="291px" height="147px" gridlines="Both">
<asp:TableRow bordercolor="Black">
<asp:TableCell font-bold="True" text="姓名"></asp:TableCell>
<asp:TableCell font-bold="True" text="成绩"></asp:TableCell>
<asp:TableCell font-bold="True" bordercolor="Transparent" text="班级"></asp:TableCell>
</asp:TableRow>
<asp:TableRow>
<asp:TableCell text="宋远桥"></asp:TableCell>
<asp:TableCell text="98"></asp:TableCell>
```



```

    <asp:TableCell text="计网 043"></asp:TableCell>
</asp:TableRow>
<asp:TableRow>
    <asp:TableCell text="张翠山"></asp:TableCell>
    <asp:TableCell text="97"></asp:TableCell>
    <asp:TableCell text="计科 044"></asp:TableCell>
</asp:TableRow>
<asp:TableRow>
    <asp:TableCell text="张无忌"></asp:TableCell>
    <asp:TableCell text="92"></asp:TableCell>
    <asp:TableCell text="计科 044"></asp:TableCell>
</asp:TableRow>
<asp:TableRow>
    <asp:TableCell text="周芷若"></asp:TableCell>
    <asp:TableCell text="60"></asp:TableCell>
    <asp:TableCell text="计科 044"></asp:TableCell>
</asp:TableRow>
</asp:Table>

```

从上述代码可以看出，整张表是包含在<asp:Table>和</asp:Table>标记之间，而表的每一行就是一对<asp:TableRow>和</asp:TableRow>标记，表单元则由<asp:TableCell>和</asp:TableCell>来定义。

实际上，Web 应用程序更多时候是需要动态生成数据表，这需要由编程来实现，这也是Table 组件的主要功能。下面的代码同样可实现上述数据表的显示，从中不难总结出动态生成数据表的一般方法。

```

var CurRow:TableRow;
    CurCell:TableCell;
begin
    // 生成第 1 行数据（标题行）
    CurRow := TableRow.Create;
    table1.Rows.Add(CurRow); // table1 为已经创建了的 Table 对象
    CurCell := TableCell.Create;
    CurRow.Cells.Add(CurCell);
    CurCell.Font.Bold := true;
    CurCell.Text:='姓名';
    CurCell := TableCell.Create;
    CurRow.Cells.Add(CurCell);
    CurCell.Font.Bold := true;
    CurCell.Text:='成绩';
    CurCell := TableCell.Create;
    CurRow.Cells.Add(CurCell);
    CurCell.Font.Bold := true;
    CurCell.Text:='班级';

    // 生成第 2 行数据
    CurRow := TableRow.Create;

```

```
table1.Rows.Add(CurRow);  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='宋远桥';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='98';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='计网 043';
```

// 生成第 3 行数据

```
CurRow := TableRow.Create;  
table1.Rows.Add(CurRow);  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='张翠山';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='97';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='计科 044';
```

// 生成第 4 行数据

```
CurRow := TableRow.Create;  
table1.Rows.Add(CurRow);  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='张无忌';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='92';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='计科 044';
```

// 生成第 5 行数据

```
CurRow := TableRow.Create;  
table1.Rows.Add(CurRow);  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='周芷若';  
CurCell := TableCell.Create;  
CurRow.Cells.Add(CurCell);  
CurCell.Text:='60';
```

```

CurCell := TableCell.Create;
CurRow.Cells.Add(CurCell);
CurCell.Text:='计科 044';
end;

```

12.4.6 HyperLink 和 LinkButton 组件

具有超链接功能是 Web 页最显著的特点之一。HyperLink 和 LinkButton 是 ASP.NET 应用程序中用来创建超链接的两个常用组件。下面分别介绍它们常用的主要属性。

HyperLink 组件的主要属性如下:

(1) Text 属性。Text 属性用于设置超链接的显示文本。

(2) ImageUrl 属性。ImageUrl 属性用于设置超链接的显示图片, 这时图片将覆盖 Text 属性设置的文本。

(3) NavigateUrl 属性。NavigateUrl 属性用于设置要链接对象的 Url 地址。Url 根据指向对象的不同可以实现不同的功能, 如打开 Web 页、下载等。

LinkButton 组件与 HyperLink 组件在外观上几乎是完全一样的, 但它们的属性却相差很大。仔细观察可以发现, LinkButton 组件的属性与 Button 组件的属性是一样的。实际上, LinkButton 对象是具有超链接功能的按钮, 而 HyperLink 对象是“纯粹”的超链接。所以, 要使用 LinkButton 来实现超链接, 需要通过简单的编程, 一般是调用 Response.Redirect 方法。

下面就通过一个例子来熟悉 HyperLink 和 LinkButton 组件的运用。

首先, 在 Delphi 2005 IDE 中创建一个 ASP.NET 应用程序, 然后分别在窗体页中放入三个 HyperLink 和三个 LinkButton 组件, 并设置它们的属性, 结果如表 12.4 所示。

表 12.4 HyperLink 和 LinkButton 组件属性的设置

组件类别	组件名称 (ID 属性值)	属性设置项目	设置结果
HyperLink	HyperLink1	Text	搜狐网站(HL)
		NavigateUrl	http://www.sohu.com
	HyperLink2	Text	联系我们(HL)
		NavigateUrl	mailto:mengzuqiang@sohu.com
	HyperLink3	Text	下载 Mp3 文件(HL)
		NavigateUrl	Mp3 文件.rar
LinkButton	LinkButton1	Text	新浪网(LB)
	LinkButton2	Text	联系我们(LB)
	LinkButton3	Text	下载 Mp3 文件(LB)

接着设计 Web 页界面, 如图 12.14 所示。

最后, 在设计界面中双击“新浪网(LB)”超链接按钮, 为该按钮添加事件处理代码如下:

```

procedure TWebForm1.LinkButton1_Click(sender: System.Object; e: System.EventArgs);
begin
    Response.Redirect('http://www.sina.com'); //链接新浪网
end;

```



图 12.14 Web 页设计界面

用同样的方法，分别为“联系我们(LB)”和“下载 Mp3 文件”超链接按钮编写事件处理代码，结果分别如下：

```
procedure TWebForm1.LinkButton2_Click(sender: System.Object; e: System.EventArgs);  
begin
```

```
    Response.Redirect('mailto:mengzuqiang@sohu.com'); //调用 Outlook 发送邮件  
end;
```

```
procedure TWebForm1.LinkButton3_Click(sender: System.Object; e: System.EventArgs);  
begin
```

```
    Response.Redirect('Mp3 文件.rar'); //下载指定的对象  
end;
```

代码编写完以后，运行该程序。经过测试可以发现，同一行上的超链接的功能完全一样，而在水平方向上从上到下各超链接的功能依次是链接新的 Web 页、打开 Outlook 邮件发送对话框及下载文件等。图 12.15 所示为单击超链接“下载 Mp3 文件”后弹出的下载界面。

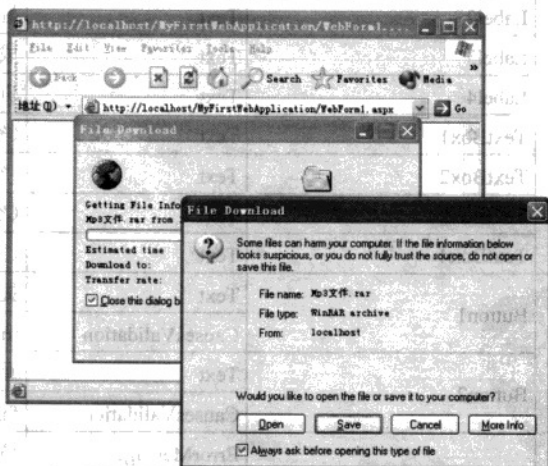


图 12.15 HyperLink 和 LinkButton 组件的运用

12.4.7 RegularExpressionValidator、RequiredFieldValidator、 RequiredFieldValidator 和 ValidationSummary 组件

由前面介绍的 Button 组件属性可知，它的 CausesValidation 属性可确定在单击该按钮时是否要进行页面验证。默认值为 true，表示需要进行验证，为 false 时则不需进行验证。

如果要进行页面验证,除了要把按钮的 CausesValidation 属性值设置为 true 外,还要创建相应的数据验证对象。RegularExpressionValidator、RequiredFieldValidator、RequiredFieldValidator 和 ValidationSummary 组件就是常用于页面验证的四个组件。对于前四个组件,它们共同拥有的四个重要属性是:

- Text 属性。用于设置当验证失败时组件显示的提示信息,默认值为空值。
- ErrorMessage 属性。ErrorMessage 属性用于设置当验证失败时组件显示的提示信息(前提是:Text 属性不被设置,即为空值,否则显示 Text 属性值)以及在 ValidationSummary 组件中显示的提示信息。
- ControlToValidate 属性。用于指定验证的对象,即设置为输入验证对象的 ID。
- EnableClientScript 属性。该属性值为布尔类型,默认值为 true,表示启用客户端验证,否则不启用。

ValidationSummary 组件需要设置的属性是 HeaderText,它用于设置组件显示错误信息的标题。在程序运行时该组件可搜集所有的错误信息并显示。

下面通过一个具体页面验证的例子来说明这些组件在验证中的使用方法。首先创建一个 ASP.NET 应用程序,然后在窗体设计页中加入四个 Label 组件,四个 TextBox 组件,两个 Button 组件,以及 RegularExpressionValidator、RangeValidator 和 ValidationSummary 组件各一个,RequiredFieldValidator 组件两个。对它们的属性设置如表 12.5 所示。

表 12.5 各组件属性的设置

组件类别	组件名称 (ID 属性值)	属性设置项目	设置结果
Label	Label1	Text	学号
	Label2	Text	姓名
	Label3	Text	成绩
	Label4	Text	班级
TextBox	TextBox1	Text	(空值)
	TextBox2	Text	(空值)
	TextBox3	Text	(空值)
	TextBox4	Text	(空值)
Button	Button1	Text	提 交
		CausesValidation	true
	Button2	Text	重 置
		CausesValidation	false
RegularExpressionValidator	RegularExpressionValidator1	ErrorMessage	请输入学号 (10 位数字)
		ControlToValidate	TextBox1
RangeValidator	RangeValidator1	ErrorMessage	请输入成绩 (0~100)
		ControlToValidate	TextBox3
ValidationSummary	ValidationSummary1	HeaderText	您必须正确填写下列选项: ● 错误信息 1 ● 错误信息 2

续表

组件类别	组件名称 (ID 属性值)	属性设置项目	设置结果
RequiredFieldValidator	RequiredFieldValidator1	ErrorMessage	请输入姓名
		ControlToValidate	TextBox2
	RequiredFieldValidator2	ErrorMessage	请输入班级信息
		ControlToValidate	TextBox4

然后对页面布局进行设计，结果如图 12.16 所示。



图 12.16 Web 页面设计结果

“重置”按钮的功能就是清除各文本框中的内容，所以对它编写如下事件处理代码：

```
TextBox1.Text="";
TextBox2.Text="";
TextBox3.Text="";
TextBox4.Text="";
```

最后运行程序，在文本框中随意输入一些值来测试，并单击“提交”按钮。如果验证通过，那么在页面中将没有任何错误提示信息；如果验证没通过，则将有相应的错误提示信息，如图 12.17 所示。



图 12.17 数据验证

12.5 熟悉 Web Form 的指令和语法

12.5.1 Web Form 指令

Web Form 指令指定在页和用户控件编译器处理 ASP.NET Web 窗体页 (.aspx) 和用户控件 (.ascx) 文件时使用的设置, 如会话状态、编码要求等。Web 窗体指令一共有八条, 现对其说明如表 12.6 所示。

表 12.6 Web 窗体指令说明

指令	作用
@ Page	定义 ASP.NET 页分析器和编译器使用的页的特定属性。该指令只能包含在.aspx 文件之中
@ Control	定义 ASP.NET 页分析器和编译器使用的控件特定属性。该指令只能包含在 .ascx 文件之中
@ Assembly	在编译时将一个程序集链接到当前页或用户控件中
@ Implements	通过该指令指示页或用户控件实现指定的 .NET Framework 接口
@ Import	将命名空间显式导入到页或用户控件, 以调用命名空间中的类和接口
@ OutputCache	通过该指令控制 ASP.NET 页或页中用户控件的输出缓存策略
@ Reference	通过该指令将指定的 ASP.NET 页或页中用户控件链接到当前页或用户控件中, 这是在编译过程中完成的
@ Register	将别名、命名空间和类名相关联起来, 以便在用户控件和自定义服务器控件语法中使用简明的表达式

12.5.2 Web Form 语法

1. 指令语法

.aspx 文件是 Web 窗体文件之一, 在本质上它是由 HTML 标签和有关控制标签组成的文本文件。这些标签的构成是按一定的规则进行的, 而这些规则就是所谓的 Web Form 语法。上一节已经对八条 Web Form 指令的作用作了说明, 下面是它们的语法格式:

<%@ 指令 属性1 = 值1 [属性2 = 值2 ...] %>

其中比较常用的是 @Page 指令, 它涵盖了原来 ASP 中的全部属性, 但 @ Page 指令只能在 .aspx 文件中使用。例如:

```
<%@ Page Language="c#" Debug="true" Codebehind="WebForm1.pas"
    AutoEventWireup="false" Inherits="WebForm1.TWebForm1" %>
```

而 @Control、@Import 等指令则是 ASP.NET 中新添加的。

2. 声明代码语法

在 ASP 中, 函数可以在 <% %> 块中声明; 但 ASP.NET 则不一样, 它的所有函数和全局页变量都必须在 <script runat=server> 标记中声明, 而 <% %> 块内声明的函数将出现编译错误。下面的示例说明如何在 <script runat=server> 块内声明 max2 的方法, 然后说明如何从页中调用此方法。

```

<html>
  <script language="C#" runat=server>
    int max2(int x, int y) {
      if x>y return x;
      return y;
    }
  </script>
  <%
    int maxNum;
    int i = 0;
    while (i<10) {
      maxNum = max2(i,10-i);
      Response.Write("值: " + number + "<br>");
      i++;
    }
  %>
</html>

```

在.aspx 文件中每一标记都有 runat 属性, 它用于指定代码是在客户端运行还是在服务器端运行, 如果是在服务器端运行则应设置为:

```
runat="server"
```

例如:

```

<asp:TextBox id="TextBox1"
  style="Z-INDEX: 12; LEFT: 142px; POSITION: absolute;
  TOP: 38px" runat="server">
</asp:TextBox>

```

3. 代码块语法

与 ASP 中的代码块语法一样, ASP.NET 中的代码块由<% ...%>元素来表示, 即在其中嵌入的代码块需要包含在<%和%>之间, 代码块中所采用的编程语言由@Page 指令中的 Language 属性指定, 默认是使用 C#语言。这些代码是在 Web 窗体页执行的呈现阶段执行。例如:

```

<form runat="server">
  <% for (int i=0; i<8; i++) { %>
    <%=i%><br>
  <% } %>
</form>

```

或

```

<form runat="server">
  <% for (int i=0; i<8; i++) { Response.Write(i);%><br><% } %>
</form>

```

这两端代码都是在 Web 页中分行打印 0~8 这九个数字。

4. 服务器控件语法

在.aspx 文件中, 用户可以根据需要自己定义 ASP.NET 服务器控件。定义方法是在文件内用基于标记的声明语法来表示。下面的示例说明如何在 ASP.NET 页内使用<asp:label runat="server">服务器控件。

首先创建 ID 为 MyLabel 的 Label 实例:

```
<asp:label id="MyLabel" font-size=24 runat="server"/>
```


然后调用或访问该控件:

```
MyLabel.Text = "Hello World!";
```

定义的该控件对应于 System.Web.UI.WebControls 命名空间中的 Label 类(默认情况下包括该命名空间)。

5. 服务器端包含语法

在服务器端可以使用 #includes 指令将指定文件的原始内容插入 ASP.NET 页内的任意位置, 其格式为:

```
<!-- #Include File="文件名" -->
```

6. 服务器端注释语法

几乎所有的编辑器都支持代码注释, 在 ASP.NET 服务器端中采用下列注释格式:

```
<%--被注释字符串--%>
```

12.5.3 ASP.NET 内置对象

ASP.NET 内置对象与 ASP 内置的对象一样, 主要包含 Page、Request、Response、Server、Application 和 Session。所不同的是, 在 ASP.NET 中, 这些对象是在 System.Web 命名空间中重新定义的类中, 它们是由 ASP.NET 自动创建的, 所以不必实例化就可以直接使用。

以下简单介绍这几个对象的作用和使用方法。

(1) Request 对象。Request 对象提供对当前页请求的访问, 常用的功能是获取客户端提交的数据, 即读取浏览器已经发送的内容。下面的代码说明了 Request 对象的一些使用方法。

```
nameStr = Request("name");  
nameStr = Request.Form("name");  
nameStr = Request.Params["name"]%
```

该对象是 HttpRequest 类的实例, 在用户请求页面时由 ASP.NET 自动产生。

(2) Response 对象。Response 对象主要用于对当前页的输出流的访问。该对象是 HttpResponse 类的实例, 在用户请求页面时由 ASP.NET 自动产生。从下面的代码中可以了解 Response 对象的一些常用方法:

```
Response.Write("中华人民共和国万岁!"); //打印输出字符串  
Response.Redirect("WebForm2.aspx"); //重定向到 WebForm2.aspx 页面  
Response.End; //结束网页的输出和服务器端脚本程序的运行
```

(3) Session 对象。Session 对象在一个 Web 会话期内有效, 通常用来保存用户等信息。Session 对象是 HttpSessionState 类的实例。下面是其常用的访问格式:

```
<%  
Session["MyName"] = "张无忌";  
Session["AppStartTime"] = DateTime.Now;  
%>  
<%= Session["MyName"]%> <br>  
<%=Session["AppStartTime"]%>
```

(4) Application 对象。Application 对象是个应用程序级的对象, 它提供了在所有用户间进行信息共享的机制, 并可以在 Web 应用程序运行期间持久地保持数据(注: 重启 IIS 后, Application 对象恢复初值, 保存的数据将丢失)。Application 对象是 HttpApplicationState 类的实例。

它具有与 Session 对象相同的用法, 如:

```

<%
Application["MyName"] = "张无忌";
Application["AppStartTime"] = DateTime.Now;
Response.Write(Application["MyName"].ToString());
%><br>
<%=Application["AppStartTime"]%>

```

由于 Application 对象是为多个程序共享的, 所以控制程序对它访问的一致性很重要。Application 对象提供的两个方法正是基于此目的, 这两个方法是 Lock 方法和 Unlock 方法。例如:

```

Application.Lock();
n = int.Parse(Application["visited_sum"].ToString());
n = n + 1;
Application["visited_sum"] = n.ToString();
Application.Unlock();

```

12.6 设计 Web Form

Web Form 的设计是指页面上各元素的大小、位置、字体、颜色等属性的协调和搭配的过程, 实际上就是网页设计。网页设计本身就是一项艰巨的工作, 它需要设计人员具有较好的审美观和符合时代特征的创意设计思维。所以, 在此不可能也没有必要对此作深入的介绍, 但作为 ASP.NET 开发人员, 掌握基本的 ASP.NET 网页设计方法是很必要的。

12.6.1 Web Form 设计环境

当创建 ASP.NET 应用程序后, Delphi 2005 IDE 就自动编制了 Web Form 设计环境, 如图 12.18 所示。当然也可以进行代码设计, 这里主要强调界面设计。



图 12.18 Web Form 设计环境

设计工作主要是在 Web 页面编辑器中完成。其中：对象观察器用于设置组件的属性；工具面板则为 Web 页面提供各种组件；代码编辑器主要是完成代码的编写任务；工程管理器则用于管理应用程序中所有的文件，包括删除或添加操作等；编辑工具栏位于 Web 页面编辑器的正上面，它提供了一些常用的编辑快捷命令按钮。

12.6.2 设置页面字体属性

1. 设置字体颜色

首先选择待设置的字，然后单击编辑工具栏上的 Font foreground color 按钮 ，打开如图 12.19 所示的 Color Selector 对话框。

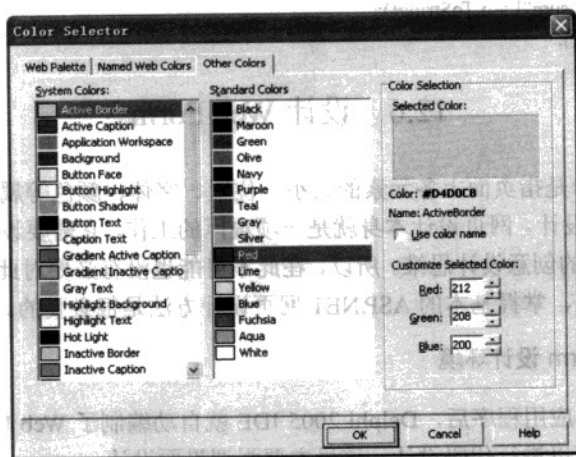





图 12.19 Color Selector 对话框

在此对话框中有三个选项卡，在每一个选项卡上都可以设置字体颜色。选择完后单击 OK 按钮，设置完毕。

2. 设置背景色

与设置字体颜色的方法类似，首先选择待设置的字，然后单击编辑工具栏上的 Font background color 按钮 ，也打开如图 12.19 所示的 Color Selector 对话框，根据需要选择相应的颜色，然后单击 OK 按钮即可。



3. 上标和下标

上标和下标是经常用到的，在此设置非常容易，方法是：选中待设置的字符，然后单击编辑工具栏上的 Superscript 按钮 ，则该字符变为上标；如果希望它被设为下标，只要单击 Subscript 按钮  即可。

4. 其他属性的设置

除了颜色和背景色外，还有字体型号、字体大小、对齐方式、是否采用粗体、是否采用斜体或下划线等，但这些设置与在 Word 中字体的设置一样，比较容易掌握，此处不再赘述。

图 12.20 所示是对字体设置的一个效果。

单击编辑工具栏上的 Show tag glyphs 按钮 ，将以相应的图标方式显示页面标签（凹状显示）；如果单击 Show grid 按钮  则会隐藏页面编辑器中的网格（凹状显示）。

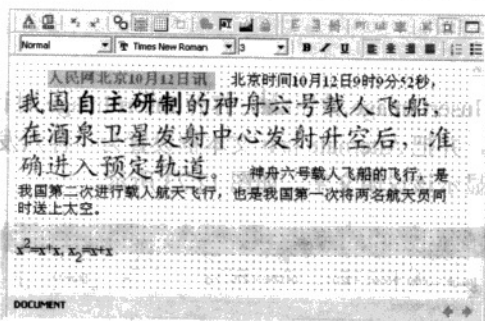


图 12.20 字体设置效果

12.6.3 使用表格

前面已经介绍了用 Table 组件来制作表格,但这种方法是为了在程序运行时动态显示数据,主要方便编程;如果静态显示数据时也使用 Table 组件,则降低设计效率。

Web Form 设计环境提供了一种简便的静态数据表格制作方法。例如,如果要制作如表 12.3 所示的数据表,可按照下列步骤完成。

(1) 单击编辑工具栏上的 Insert table 按钮, 打开 Insert Table 对话框,然后把 Rows 值设为“5”,Columns 设为“3”,如图 12.21 所示,单击 OK 按钮。如果需要还可以进行其他设置。

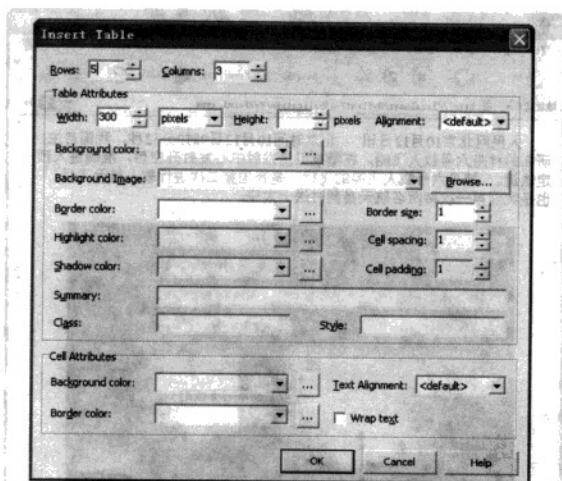




图 12.21 Insert Table 对话框

(2) 把表 12.3 中的数据分别输入到对象的表格单元中,然后选择第一行数据,单击编辑工具栏上的 Bold 按钮,把第一行数据变为黑体字,结果如图 12.22 所示。

姓名	成绩	班级
宋远桥	98	计网043
张翠山	97	计科044
张无忌	92	计科044
周芷若	60	计科044

图 12.22 在 Web 页面编辑器中创建表格

12.6.4 插入图片

单击编辑工具栏上的 Insert Image 按钮，打开 Insert Image 对话框，然后单击 Browse... 按钮，选择要添加的图片，并把 Alternate text 文本框设置为“神六发射瞬间”，在浏览时鼠标停留在图片上方片刻即可显示该说明文字，如图 12.23 所示。

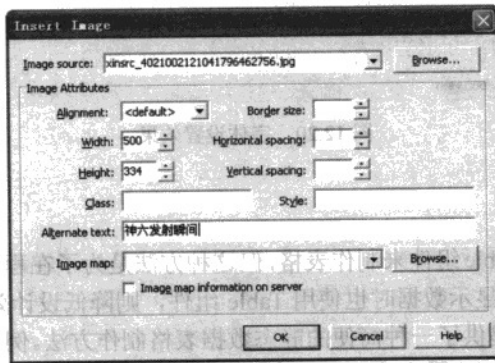


图 12.23 Insert Image 对话框


单击 OK 按钮，图片加入完毕。运行程序，结果如图 12.24 所示。




图 12.24 在 Web 页面中加入图片

12.6.5 创建超链接

创建超链接当然可以用前面介绍的 HyperLink 和 LinkButton 组件来完成，但是这两个组件的运用也主要是面向编程的，而对于静态的文字超链接则不适宜。实际上，静态文本文字的超链接通常用下列方法来完成。

(1) 选择待创建超链接的文本文字, 这时编辑工具栏上的 Create link 按钮变为有效状态 (选择前是灰色)。

(2) 单击 Create link 按钮, 将弹出如图 12.25 所示的 Hyperlink 对话框。在 Type 下拉列表框中选择超链接的类型, 如下载、打开 Outlook 邮件发送对话框等, 然后在 URL 文本框中输入要链接的对象, 最后单击 OK 按钮, 超链接创建完毕。

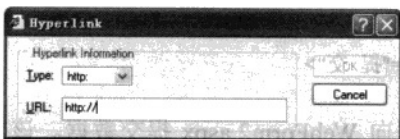


图 12.25 Hyperlink 对话框

12.7 执行页导航及参数传递

在 Web 应用程序中, 实现多页之间的页面导航是最基本的功能之一, 也是非常重要的任务, 如注册页、提交页等。其中的一个难点就是, 如何有效实现页面之间的参数传递。本节将根据不同的参数传递方式来介绍几种页导航方法。

12.7.1 使用 Form 传递 (提交) 数据

Form 组件一般要与 Button 组件结合在一起使用。下面通过一个具体的例子来介绍这种方法。

首先创建一个 ASP.NET Web 应用程序, 命名为 SubmitWebSample.dpr。创建后, 程序中仅有一个.aspx 文件——WebForm1.aspx, 为此需要再添加一个.aspx 文件, 即 WebForm2.aspx。这个例子的目的就是实现由 WebForm1.aspx 向 WebForm2.aspx 传递 (提交, 请求处理)。

添加 WebForm2.aspx 文件的步骤如下:

(1) 在 Delphi 2005 IDE 中选择菜单 View→Project Manager 命令, 打开工程管理器。

(2) 在工程管理器中右击结点“SubmitWebSample.dll”, 在弹出的快捷菜单中选择 New→Other...命令, 将打开 New Items 对话框。在该对话框左边的 Item Categories 方框中选择 New ASP.NET Files 选项, 然后在右边的方框中选择 ASP.NET Page 图标。

(3) 单击 OK 按钮, 在工程管理器中会增加一个新的结点——WebForm2.aspx 结点, WebForm2.aspx 正是刚创建的.aspx 文件。

(4) 编辑 WebForm1.aspx 文件, 结果如下:

```
<form action="WebForm2.aspx" method="post">
```

学号:

```
<input name="stu_id">
```

```
<br>
```

```
<br>
```

姓名:

```
<input name="name">
```

```
<br>
```

```
<br>
```

成绩:

```

<input name="grade">
<br>
<br>
班级:
<input name="class">&nbsp;   
<br>
<br>
&nbsp;   
<input type="submit" value="提 交">
</form>

```

该文件是采用 post 方法向 WebForm2.aspx 提交数据。该文件虽然是.aspx 文件，但其作用仅相当于客户端文件，所以没有 runat="server" 属性。实际上，该文件也可以改用 html 文件充当。

WebForm2.aspx 文件的代码如下：

```

<form runat="server"><%
    int stu_id;
    String name;
    float grade;
    String className;
    stu_id = int.Parse(Request.Params["stu_id"]);
    name = Request.Params["name"];
    grade = float.Parse(Request.Params["grade"]);
    className = Request.Params["class"];
%>
<%= "学号: "+stu_id%>
    <br><br>
<%= "姓名: "+name%>
    <br><br>
<%= "成绩: "+grade%>
    <br><br>
<%= "班级: "+className%>
    <br><br>
</form>

```

该文件是服务器端文件，必须是.aspx 文件，其中每一组件必须包含 runat="server" 属性。它用于对提交数据进行处理。

(5) 代码编写完成后，请务必记住保存上述两个文件，然后在 IE 中打开 WebForm1.aspx 文件，并输入相应的数据进行测试，如图 12.26 所示。

(6) 数据输入完毕以后，单击“提交”按钮，向服务器提交数据，以请求对数据的加工和处理。这个实例只是把从客户端中获得的数据进行适当的类型转换，最终将其打印出来，如图 12.27 所示。

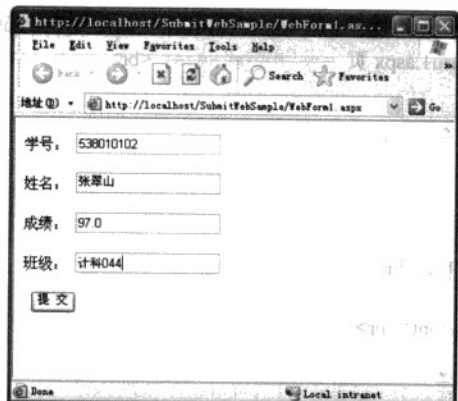


图 12.26 输入数据 (客户端)

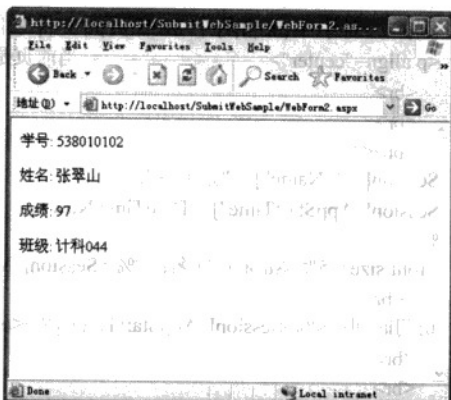


图 12.27 数据处理结果

12.7.2 使用超链接传递数据

使用超链接转向或打开新网页的同时也可以传递参数,方法是在新网页名后面加上问号“?”,然后后接“参数名=值”的形式,如果多个参数要传递,则用符号“&”隔开,并且符号“?”和“&”前后不能有空格。

例如,如果要把下面这条数据向服务器提交:

(538010102, 殷离, 78.0, 计网 043)

可用下列超链接实现 (WebForm1.aspx 文件):

```
<form action="" runat="server">
<a href="http://localhost/SubmitWebSample/WebForm2.aspx?stu_id=538010102&name=殷离
&grade=78.0&class=计网 043"> 传递参数 </a>
</form>
```

而 WebForm2.aspx 文件不变 (与上一节介绍的一样)。

12.7.3 页面重定向和参数传递

页面重定向一般是用 Response 对象的 Redirect 方法来完成的,其参数格式与使用超链接传递的情况类似。例如,把 WebForm1.aspx 对应的页面重定向到 WebForm2.aspx 对应的页面,并传递下列参数:

(538010102, 殷离, 78.0, 计网 043)

则可以使用下列代码 (WebForm1.aspx 文件):

```
<form action="" runat="server">
<%
Response.Redirect("http://localhost/SubmitWebSample/WebForm2.aspx?stu_id=538010102
&name=殷离&grade=78.0&class=计网 043");
%>
</form>
```

12.7.4 使用 Session 对象传递参数

Session 对象在一个会话期内有效,所以在一个会话期内任何一个页面都可以访问 Session 对象。可以利用这一点来实现不同页面之间参数的传递。

例如,对于下面的两个 Web 页面文件之间进行参数传递,以下是 WebForm1.aspx 文件代码。

```
<p align="center">===== 当前页是 WebForm1.aspx 页 =====<br>
<br>
<br><%
Session["MyName"] = "张无忌";
Session["AppStartTime"] = DateTime.Now;
%>
<font size="5"><strong>姓名: <%= Session["MyName"]%><br>
<br>
访问时间: <%=Session["AppStartTime"]%></strong> </font><br>
<br>
<br>
<a href="http://localhost/SubmitWebSample/WebForm2.aspx">点击此处转到 WebForm2.aspx 页</a>
</p>
```

(以下是 WebForm2.aspx 文件代码)

```
<br>
<br>
<p align="center">===== 当前页是 WebForm2.aspx 页 =====<br>
<br>
<br>
<font size="5"><strong>姓名: <%= Session["MyName"]%><br>
<br>
访问时间: <%=Session["AppStartTime"]%>
</strong>
</font><br>
<br>
<br>
</p>
```

在 WebForm1.aspx 页中对 Session["MyName"]和 Session["AppStartTime"]赋值,然后在 WebForm2.aspx 页中继续访问这两个值,发现这两个值没有改变,如图 12.28 和图 12.29 所示,这说明了在页面跳转中 Session 对象具有传递参数的作用。

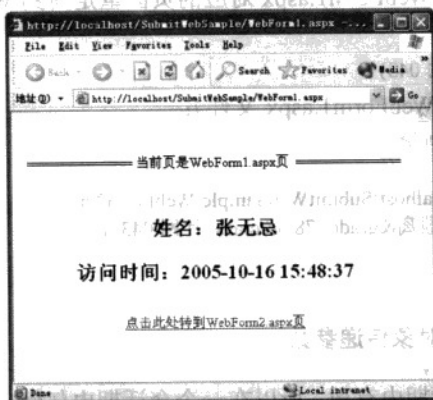


图 12.28 WebForm1.aspx 页 (设置和访问 Session 对象)



图 12.29 WebForm2.aspx 页（设置和访问 Session 对象）

使用 Session 对象会消耗内存资源，所以不宜过多使用，一般只是用于保存用户的基本信息，以便“随地”都可以访问。

12.8 ASP.NET 数据库开发

在 Delphi 2005 中，开发数据库应用程序可以有很多种方法，如利用 ADO.NET 技术，这完全可以参照第 8 章介绍的内容，在此不再重复。本节主要介绍基于 Borland 公司提供的 BDP.NET 技术的数据库开发方法。这种方法主要是利用 Borland Data Provider 标签页中的组件来实现的，现以对数据库 MyDatabase 的数据表 stu 中的数据管理为例介绍这种方法。

(1) 首先创建一个 ASP.NET 应用程序，命名为 ASPDatabaseSim.pdr，方法是：在 Delphi 2005 IDE 中选择菜单 File→New→ASP.NET Web Application - Delphi for .NET 命令，在打开的对话框中把程序名设置为 ASPDatabaseSim.dpr，然后单击 OK 按钮，一个空白的 ASP.NET 应用程序就创建完毕了。

(2) 打开工具面板，从 Borland Data Provider 标签页中把 BdpConnection 组件添加到窗体上，采用默认名 BdpConnection1 (name 属性值)；然后在对象观察器中设置 BdpConnection 组件的 ConnectionString 属性。方法是：找到该属性项，然后单击该属性项右边的下拉列表按钮选择已设置好的数据库链接；如果还没有配置好相应的数据库链接，则右击 BdpConnection 组件，在弹出的快捷菜单中选择 Connections Editor... 命令，打开数据库链接编辑 (Connections Editor) 对话框。

在此对话框中，Connections 框中的项是已经存在的数据库链接，右边方框中的内容表示对应项的详细设置情况。

(3) 单击 Add 按钮，弹出 Add New Connection 对话框，在 Provider Name 下拉列表框中选择 MSSQL，数据库链接名设置为 MyASPCon。

(4) 单击 OK 按钮，又回到数据库链接编辑对话框，然后对 MyASPCon 的以下几个属性值进行设置：

- Database: 把该属性值设置为 MyDatabase，表示 MyASPCon 是与数据库 MyDatabase 建立联系。

- **HostName:** 该属性值用于指定数据库服务器的名称, localhost 表示本地机器名。
- **OSAuthentication:** 该属性用于设置登录数据库的验证方式, 当其值为 true 时表示登录数据库要用进入 Windows 系统所需的用户名和密码, 此时 Password 和 UserName 属性无效。本例设置为 false。
- **UserName:** 登录数据库的用户名, 当 OSAuthentication 取值 true 有效。本例设置为 sa。
- **Password:** 登录数据库的密码, 当 OSAuthentication 取值 true 有效。本例设置为“123” (在数据库安装时设置的密码)。

本例设置情况如图 12.30 所示。设置完成以后, 可单击 Test 按钮来测试所创建的链接是否有效, 如果有效则单击 OK 按钮, 名为 MyASPCon 的数据库链接创建完毕, 否则逐项检查, 直到测试通过为止。

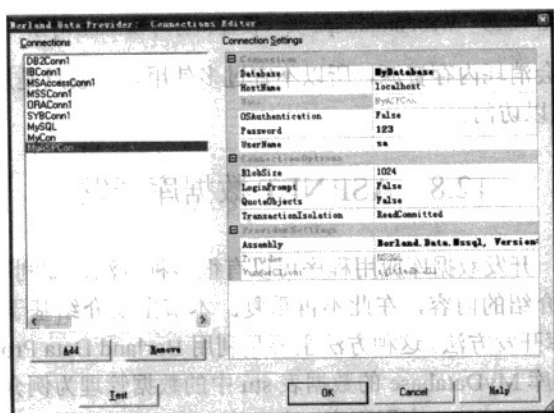


图 12.30 设置名为 MyASPCon 的数据库链接

(5) 在窗体上添加 SqlDataAdapter 组件, 然后右击该组件, 在弹出的快捷菜单中选择 Configure Data Adapter... 命令, 将打开 Data Adapter Configuration 对话框。在此对话框中需要进行以下设置:

- 在 Connection 下拉列表框中选择 BdpConnection1。

- 在 Tables 框中选择 dbo.STU 项。

- 在 Columns 框中选择 “*”，表示要查询所有的列。

- 单击 Generate SQL 按钮, 产生下列 SQL 代码:

```
SELECT STU_ID, NAME, GENDER, BIRTHDAY, NATIVE, SPECIALITY,  
GRADE, REMARK FROM dbo.STU
```

这个例子的设置结果如图 12.31 所示。

接着选择 DataSet 标签页, 在打开的页面中选择单选按钮 New DataSet, 表示要创建名为 dataSet1 的数据集对象。

(6) 单击对话框中的 OK 按钮, 对数据提供者的设置完毕。这时发现窗体中已经创建了一个 DataSet 对象。

(7) 在对象观察器中把 SqlDataAdapter 对象的 Active 属性值设置为 true, 使链接生效。至此, 数据库链接成功。为了能把数据显示出来, 并进行一些插入、删除和修改等操作,

12.9 ASP.NET 应用程序开发实例：网上商店

本节主要运用前面介绍的有关知识，开发一个 ASP.NET 程序实例。从中读者可以发现，在 Delphi 2005 中 ASP.NET 程序的开发在很大程度上变成了 Delphi 应用程序的开发，所以通过利用 Delphi 2005，很多 Delphi 程序员可以很轻松地转变成为 ASP.NET 应用程序开发人员。

12.9.1 创建 ASP.NET 应用程序及其界面设计

在 Delphi 2005 IDE 中创建一个 ASP.NET 应用程序，命名为 WebAppPurchasing.dpr。然后在窗体设计页中加入五个 Button 组件、一个 TextBox 组件、两个 ListBox 组件及五个 Label 组件，并设置它们的属性，结果如表 12.8 所示。

表 12.8 Web 页中各组件属性的设置

组件类别	组件名称 (ID 属性值)	属性设置项目	设置结果
Label	Label1	Text	当前可购买的商品:
	Label2	Text	当前已购买的商品:
	Label3	Text	添加商品:
	Label4	Text	(空值)
		Font.Bold	true
		Font.Size	18pt
		ForeColor	Red
	Label5	Text	网 上 商 店
		Font.Bold	true
		Font.Size	28pt
		ForeColor	Red
Button	Button1	Text	购买当前商品
		BorderWidth	1px
	Button2	Text	退回当前商品
		BorderWidth	1px
	Button3	Text	添 加
	Button4	Text	购买所有商品
ListBox	ListBox1	Items	初值设置如图 12.33 所示
	ListBox2	Items	(空值)
TextBox	TextBox1	Text	(空值)

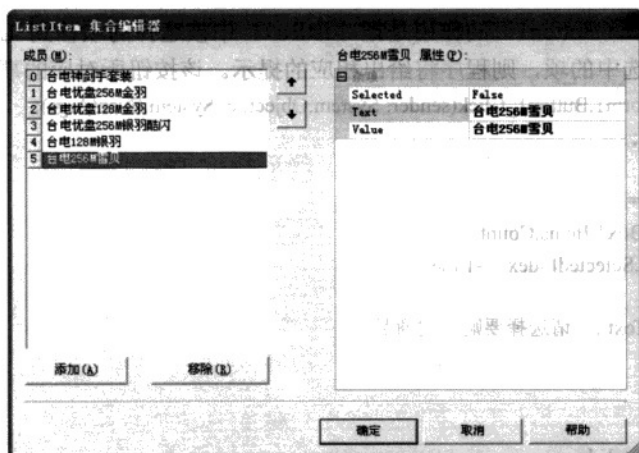


图 12.33 ListBox1 的 Items 属性值初值设置

适当调整各组件的位置和大小, 结果如图 12.34 所示。

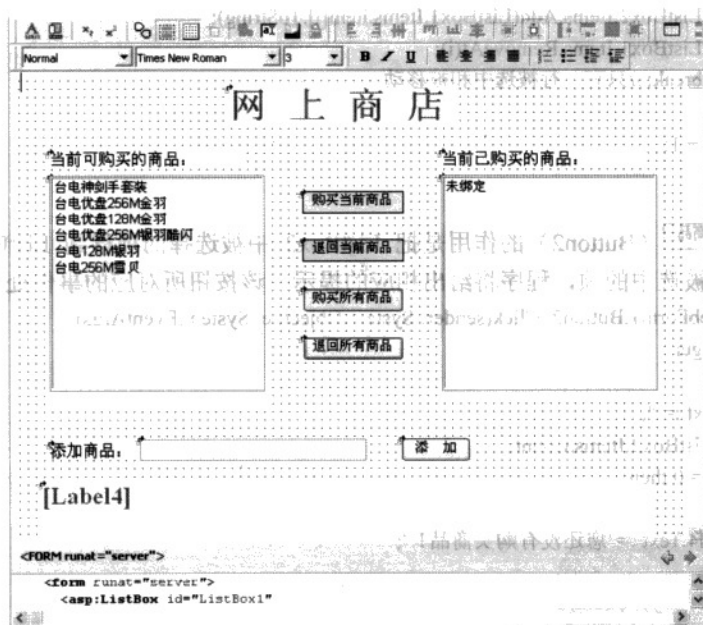


图 12.34 Web 页设计效果

12.9.2 程序代码设计

代码设计就是为各按钮编写事件处理代码。该程序一共有五个按钮组件, 在设计窗体中只要双击某一个按钮, Delphi 即可自动为之创建事件的处理过程, 同时光标将停留在过程的代码编写处。产生的代码将保存在窗体的代码文件 WebForm1.pas 中。下面是程序员为各按钮编写的事件处理代码。

按钮 **购买当前商品** (Button1) 的作用是把 ListBox1 中被选择的项移到 ListBox2 中, 如果 ListBox1 中没有被选中的项, 则程序将给出相应的提示。该按钮所对应的事件处理代码如下:

```
procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox1.Items.Count;
    if ListBox1.SelectedIndex = -1 then
    begin
        Label4.Text := '请选择要购买的商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        if ListBox1.Items.item[i].Selected then
        begin
            ListBox2.Items.Add(ListBox1.Items.item[i].ToString);
            ListBox1.Items.RemoveAt(i);
            break; //只有一行被选中和被移动
        end;
        i := i + 1;
    end;
end;
```

按钮 **退回当前商品** (Button2) 的作用是把 ListBox2 中被选择的项移到 ListBox1 中, 如果 ListBox2 中没有被选中的项, 程序将给出相应的提示。该按钮所对应的事件处理代码如下:

```
procedure TWebForm1.Button2_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox2.Items.Count;
    if count = 0 then
    begin
        Label4.Text := '您还没有购买商品!';
        exit;
    end;
    if ListBox2.SelectedIndex = -1 then
    begin
        Label4.Text := '请选择要退回的商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        if ListBox2.Items.item[i].Selected then
        begin
```

```

        ListBox1.Items.Add(ListBox2.Items.item[i].ToString);
        ListBox2.Items.RemoveAt(i);
        break; //只有一行被选中和被移动
    end;
    i := i + 1;
end;
end;

```

按钮 (Button4) 的作用是把 ListBox1 中所有的项 (不管是否被选中) 移到 ListBox2 中, 如果 ListBox1 中没有被选中的项, 则程序将给出相应的提示。该按钮所对应的事件处理代码如下:

```

procedure TWebForm1.Button4_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox1.Items.Count;
    if count = 0 then
    begin
        Label4.Text := '您已经购买了所有商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        ListBox2.Items.Add(ListBox1.Items.item[i].ToString);
        i := i + 1;
    end;
    ListBox1.Items.Clear;
end;

```

按钮 (Button5) 的作用是把 ListBox2 中所有的项 (不管是否被选中) 移到 ListBox1 中, 如果 ListBox2 中没有被选中的项, 则程序将给出相应的提示。该按钮所对应的事件处理代码如下:

```

procedure TWebForm1.Button5_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox2.Items.Count;
    if count = 0 then
    begin
        Label4.Text := '您还没有购买商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        ListBox1.Items.Add(ListBox2.Items.item[i].ToString);
    end;
    end;
end;

```



```

        i := i + 1;
    end;
    ListBox2.Items.Clear;
end;

```

经过以上的代码编写过程后，就可以运行该程序了。但为了方便读者的比较和学习，本文给出了文件 WebForm1.aspx 和 WebForm1.pas 的完整代码。这两个文件的代码分别如下：

```

//////////////////////////////////// WebForm1.aspx //////////////////////////////////////
<form runat="server">
<asp:ListBox id="ListBox1"
    style="Z-INDEX: 5; LEFT: 38px; POSITION: absolute; TOP: 102px"
    runat="server" rows="3" width="211px" height="227px">
    <asp:ListItem value="a">台电神剑手套装</asp:ListItem>
    <asp:ListItem value="b">台电优盘 256M 金羽</asp:ListItem>
    <asp:ListItem value="c">台电优盘 128M 金羽</asp:ListItem>
    <asp:ListItem value="d">台电优盘 256M 银羽酷闪</asp:ListItem>
    <asp:ListItem value="台电 128M 银羽">台电 128M 银羽</asp:ListItem>
    <asp:ListItem value="台电 256M 雪贝">台电 256M 雪贝</asp:ListItem>
</asp:ListBox>
<asp:Label id="Label1" style="Z-INDEX: 6; LEFT: 38px; POSITION: absolute; TOP: 78px"
    runat="server" width="187px">当前可购买的商品: </asp:Label>
<asp:Label id="Label3" style="Z-INDEX: 6; LEFT: 38px; POSITION: absolute; TOP: 366px"
    runat="server" width="83px">添加商品: </asp:Label>
<asp:Label id="Label2" style="Z-INDEX: 7; LEFT: 422px; POSITION: absolute; TOP: 78px"
    runat="server" width="195px">当前已购买的商品: </asp:Label>
<asp:ListBox id="ListBox2"
    style="Z-INDEX: 8; LEFT: 422px; POSITION: absolute; TOP: 102px"
    runat="server" width="211px" height="219px"></asp:ListBox>
<asp:Button id="Button1"
    style="Z-INDEX: 9; LEFT: 286px; POSITION: absolute; TOP: 118px"
    runat="server" width="99px" borderwidth="1px" text="购买当前商品"></asp:Button>
<asp:Button id="Button2"
    style="Z-INDEX: 10; LEFT: 286px; POSITION: absolute; TOP: 166px"
    runat="server" width="99px" borderwidth="1px" text="退回当前商品"></asp:Button>
<asp:TextBox id="TextBox1"
    style="Z-INDEX: 11; LEFT: 126px; POSITION: absolute; TOP: 362px"
    runat="server" width="222px"></asp:TextBox>
<asp:Button id="Button3"
    style="Z-INDEX: 12; LEFT: 382px; POSITION: absolute; TOP: 362px"
    runat="server" width="68px" text=" 添 加 "></asp:Button>
<asp:Button id="Button4"
    style="Z-INDEX: 13; LEFT: 286px; POSITION: absolute; TOP: 214px"
    runat="server" width="99px" text="购买所有商品"></asp:Button>
<asp:Button id="Button5"
    style="Z-INDEX: 14; LEFT: 286px; POSITION: absolute; TOP: 262px"
    runat="server" width="99px" text="退回所有商品"></asp:Button>
<asp:Label id="Label4" style="Z-INDEX: 15; LEFT: 30px; POSITION: absolute; TOP: 406px"

```

```

        runat="server" width="595px" height="54px" font-bold="True"
        font-size="18pt" forecolor="Red"></asp:Label>
<asp:Label id="Label5" style="Z-INDEX: 16; LEFT: 214px; POSITION: absolute; TOP: 14px"
        runat="server" width="222px" font-bold="True" font-size="28pt"
        forecolor="Red">网 上 商 店</asp:Label>
</form>

```

```

//////////////////////////////////// WebForm1.pas //////////////////////////////////////

```

```

unit WebForm1;
interface
uses
    System.Collections, System.ComponentModel, SysUtils,
    System.Data, System.Drawing, System.Web, System.Web.SessionState,
    System.Web.UI, System.Web.UI.WebControls, System.Web.UI.HtmlControls;
type
    TWebForm1 = class(System.Web.UI.Page)
    {$REGION 'Designer Managed Code'}
    strict private
        procedure InitializeComponent;
        procedure Button3_Click(sender: System.Object; e: System.EventArgs);
        procedure Button1_Click(sender: System.Object; e: System.EventArgs);
        procedure Button2_Click(sender: System.Object; e: System.EventArgs);
        procedure Button4_Click(sender: System.Object; e: System.EventArgs);
        procedure Button5_Click(sender: System.Object; e: System.EventArgs);
    {$ENDREGION}
    strict private
        procedure Page_Load(sender: System.Object; e: System.EventArgs);
    strict protected
        ListBox1: System.Web.UI.WebControls.ListBox;
        Label1: System.Web.UI.WebControls.Label;
        Label2: System.Web.UI.WebControls.Label;
        ListBox2: System.Web.UI.WebControls.ListBox;
        Button1: System.Web.UI.WebControls.Button;
        Button2: System.Web.UI.WebControls.Button;
        TextBox1: System.Web.UI.WebControls.TextBox;
        Label3: System.Web.UI.WebControls.Label;
        Button3: System.Web.UI.WebControls.Button;
        Button4: System.Web.UI.WebControls.Button;
        Button5: System.Web.UI.WebControls.Button;
        Label4: System.Web.UI.WebControls.Label;
        Label5: System.Web.UI.WebControls.Label;
        procedure OnInit(e: EventArgs); override;
    private
        { Private Declarations }
    public
        { Public Declarations }

```

```

end;
implementation
{$REGION 'Designer Managed Code'}

/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWebForm1.InitializeComponent;
begin
    Include(Self.Button1.Click, Self.Button1_Click);
    Include(Self.Button2.Click, Self.Button2_Click);
    Include(Self.Button3.Click, Self.Button3_Click);
    Include(Self.Button4.Click, Self.Button4_Click);
    Include(Self.Button5.Click, Self.Button5_Click);
    Include(Self.Load, Self.Page_Load);
end;
{$ENDREGION}

procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
begin
    // TODO: Put user code to initialize the page here
end;

procedure TWebForm1.OnInit(e: EventArgs);
begin
    //
    // Required for Designer support
    //
    InitializeComponent;
    inherited OnInit(e);
end;

procedure TWebForm1.Button5_Click(sender: System.Object; e: System.EventArgs);
var i, count: integer;
begin
    Label4.Text := "";
    count := ListBox2.Items.Count;
    if count = 0 then
        begin
            Label4.Text := '您还没有购买商品!';
            exit;
        end;
    i:=0;
    while i<=count-1 do
        begin

```

```
        ListBox1.Items.Add(ListBox2.Items.item[i].ToString);
        i := i + 1;
    end;
    ListBox2.Items.Clear;
end;

procedure TWebForm1.Button4_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox1.Items.Count;
    if count = 0 then
    begin
        Label4.Text := '您已经购买了所有商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        ListBox2.Items.Add(ListBox1.Items.item[i].ToString);
        i := i + 1;
    end;
    ListBox1.Items.Clear;
end;

procedure TWebForm1.Button2_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox2.Items.Count;
    if count = 0 then
    begin
        Label4.Text := '您还没有购买商品!';
        exit;
    end;
    if ListBox2.SelectedIndex = -1 then
    begin
        Label4.Text := '请选择要退回的商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        if ListBox2.Items.item[i].Selected then
        begin
            ListBox1.Items.Add(ListBox2.Items.item[i].ToString);
```

```
        ListBox2.Items.RemoveAt(i);
        break; //只有一行被选中和被移动
    end;
    i := i + 1;
end;
end;
```

```
procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    count := ListBox1.Items.Count;
    if ListBox1.SelectedIndex = -1 then
    begin
        Label4.Text := '请选择要购买的商品!';
        exit;
    end;
    i:=0;
    while i<=count-1 do
    begin
        if ListBox1.Items.item[i].Selected then
        begin
            ListBox2.Items.Add(ListBox1.Items.item[i].ToString);
            ListBox1.Items.RemoveAt(i);
            break; //只有一行被选中和被移动
        end;
        i := i + 1;
    end;
end;
```

```
procedure TWebForm1.Button3_Click(sender: System.Object; e: System.EventArgs);
var i,count:integer;
begin
    Label4.Text := "";
    if TextBox1.Text = "" then
    begin
        Label4.Text := '请输入商品名称!';
        exit;
    end;
    count := ListBox1.Items.Count;
    for i:=0 to count-1 do
    begin
        if TextBox1.Text=ListBox1.Items.item[i].ToString then
        begin
            Label4.Text := '已有该商品!';
            exit;
        end;
    end;
end;
```

```
end;  
end;  
ListBox1.Items.Add(TextBox1.Text); //为 ListBox1 添加数据  
TextBox1.Text := ""; //清空 TextBox1  
end;  
end.
```

12.9.3 程序的运行和使用

在编写上述代码以后，运行该程序。可以看到，打开运行时列表框 ListBox1 中仅有下面一段文字：

台电神剑手套装
台电优盘 256M 金羽
台电优盘 128M 金羽
台电优盘 256M 银羽酷闪
台电 128M 银羽
台电 256M 雪贝

然后，在文本框中输入“台电优盘 512M 骑士”并单击“添加”按钮，则“台电优盘 512M 骑士”被加入到“当前可购买的商品：”列表框 (ListBox1) 中。选中其中的某一项，通过“购买当前商品”按钮可以把它移到“当前已购买的商品：”列表框中 (ListBox2) 中；反之，把“当前已购买的商品：”列表框中的某一项移到“当前可购买的商品：”列表框中，则可以通过“退回当前商品”按钮实现。图 12.35 所示为其中的一个结果。当然，可以一次购买所有的商品，或者退回所有已购买的商品，其操作是显然的。

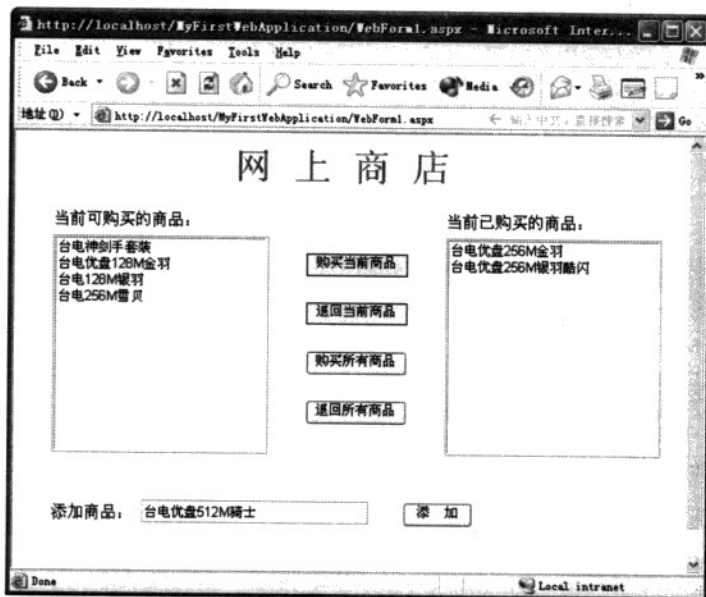


图 12.35 程序运行的一个状态

12.10 小结

本章介绍了 ASP.NET 的特点、ASP.NET 的技术，以及基于 ASP.NET 技术的应用程序开发技能。通过对本章的学习，读者应掌握下列内容：

- ASP.NET 运行环境的搭建和配置，包括 IIS 的安装和设置等。
- ASP.NET 应用程序的文件结构。
- 常用的 ASP.NET 组件的属性和方法，能够较为熟练地使用它们。
- Web Form 的指令和语法及其 ASP.NET 内置对象。
- 执行页导航及参数传递方法。
- ASP.NET 应用程序开发的一般方法。
- 基于 BDP 和 ADO.NET 的 ASP.NET 数据库开发技术。

第 13 章 熟悉 Delphi 中 ASP.NET Web Services 应用程序开发

随着 Internet 技术的迅猛发展和网上资源的日益丰富, 如何把这些资源有机地整合起来, 为特定的应用服务, 这是一个很现实的问题。Web Services (Web 服务) 正是迎合了这种需要而不断得到发展的一种热门的 Web 开发技术。本章着重介绍了利用 Delphi 开发 ASP.NET Web Services 的技术, 涉及的要点包括:

- Web 服务的基本概念。
- Web 服务的特点, 包括优缺点、体系结构和协议栈等。
- 创建 Web 服务应用程序的基本步骤及其文件结构。
- Web 服务应用的开发和访问技术。

13.1 Web 服务 (Web Services) 简介

13.1.1 什么是 Web 服务

随着 Internet 技术的诞生和迅猛发展, 微软公司越来越重视在这个领域的投资和技术开发。2001 年微软推出了具有重要战略意义的 Microsoft .NET 平台, 希望通过这个平台可以实现基于 Internet 的信息、资源和服务共享, 使得在任何时候、任何地方、利用任何工具都可以获得 Internet 上的信息和享受 Internet 提供的服务。

Microsoft .NET 已经开创了 Internet 的新局面, 基于 HTML 的显示信息将通过可编程的基于 XML 的信息得到增强。其中, Web 服务 (Web Services) 是 Microsoft.NET 和 SunONE 技术的核心, 可以从不同层次和角度来理解它。首先, Web 服务是一种软件开发思想——软件就是服务, 这已经是软件发展的一个潮流了。其次, Web 服务是一种应用程序, 它可以使用标准的互联网络协议, 如超文本传输协议 HTTP 和 XML 等, 将功能纲领性地体现在互联网和企业内部网上, 供其他应用程序使用。最后, 从开发的角度讲, Web 服务是 Web 上可编程的组件, 程序员可以像调用本地服务一样通过调用 Web 应用编程接口 (API) 来提供信息和服务。这些信息和服务是分布在 Internet 上的, 通过客户端程序把它们集成起来。这使得看起来好像是专为某个人所用, 而实际上是为大家使用, 这样就使得 Internet 成为一个软件开发平台。这正是微软推出 Microsoft .NET 的初衷。

从另一个角度看, Web 服务是一个由服务来描述的接口, 即服务是由服务描述来实现的, 服务描述包含服务的接口和实现的细节, 它可以被发布给服务请求者或服务注册中心。

13.1.2 Web 服务的优缺点

Web 服务是一种瘦客户服务程序, 当前已成为一种热门的 Web 开发技术。使用 Web 服务的优点体现在以下几个方面:

- 在 Internet 上具有丰富的资源, 通过 Web 服务可以把一些资源集成到一个应用程序中,

可以在任何地方使用它们，同时使得 Internet 成为软件开发的一个平台，这是 Web 服务的魅力所在。

- Web 服务具有客户/服务器 (C/S) 模型的优点，即绝大部分的处理都在服务器上完成，采用瘦客户端技术（多用 Web 浏览器，不用任何的安装），可以有效地避免因用于发布桌面应用程序所需的高成本。
- Web 服务具有跨平台的特性，其采用 HTTP 和 SOAP 协议等都是 Internet 上通用的传输协议，这是它具有跨平台特性的重要保证；而且 SOAP 协议可以“穿越”防火墙，这有效地保证了使用 Web 服务可以通过防火墙进行通信，而基于 DCOM 或 CORBA 技术的访问则被“挡”在防火墙外面。
- Web 服务可以很容易地实现通过 Internet 进行操作或远程调用，而且使得操作和远程调用更具功效性，特别是在进行跨公司的商务交易和 B2B 集成时。

Web 服务是基于 Internet 的应用程序，如果把它限制在单机上或限制在局域网中，由于其消耗资源大而导致效率降低，这时不如使用基于 DCOM 的客户/服务器模型来得更为有效。

13.1.3 Web 服务体系结构

Web 服务体系结构的主要特点之一就是，允许在不同平台上基于同一种标准协议使用不同编程语言来开发应用程序。它不仅允许两个程序通过 Internet 来通信，还可以允许将多个 Web 服务整合到一个用户定制的应用程序中。图 13.1 给出了 Web 服务的面向 Service 的体系结构。

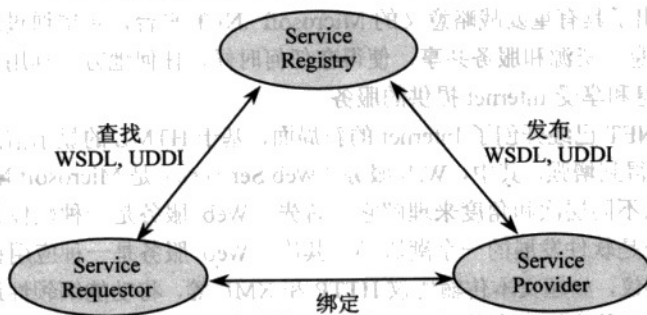


图 13.1 Web 服务的体系结构

在 Web 服务体系结构中，主要的角色是 Service Registry（服务注册中心）、Service Requestor（服务请求者）和 Service Provider（服务提供者）。它们的作用如下：

- Service Provider（服务提供者）：用于发布自己的服务，并对服务请求进行响应。从体系结构的角度看，这是托管访问服务的平台；从使用者的角度看，这是服务的所有者。
- Service Registry（服务注册中心）：用于注册已经发布的服务（Services），并提供搜索服务的功能。从体系结构的角度看，这是寻找并调用服务、启动与服务交互的应用程序。
- Service Requestor（服务请求者）：是服务的使用者。它要通过 Service Registry 来查找服务，然后由 Web 浏览器、另一个 Web 服务或者瘦客户端程序来控制和使用该服务。

在 Web 服务体系结构中，如果一个服务是可访问的，则必须先进行下列操作：

- 发布服务描述：其目的是使服务“活动”起来，以便服务请求者可以查找它，然后使用它。
- 查找服务：服务请求者直接检索服务描述，或者在服务注册中心查询所要求的服务类型。
- 绑定：服务请求者通过使用服务描述中的绑定细节来定位、联系和调用服务，从而在运行时调用或启动与服务的交互程序。

从图 13.1 中还可以看出，服务请求者和服务注册中心、服务提供者和服务注册中心之间的交互是通过利用 UDDI 和 WSDL 协议来实现的，而它们又建立在 HTTP 基础之上。实际上，Web 服务是由一系列开放的 Internet 协议构成的协议栈来支撑其运行的。所以，要了解 Web 服务体系的运行机制，还需了解赖以存在的协议栈。

13.2 熟悉 Web 服务协议栈

Web 服务是基于开放的 Internet 标准，这些标准就构成了 Web 服务协议栈。下面对各层协议分别进行说明。

13.2.1 SOAP

上层协议是建立在下层提供的功能之上，如图 13.2 所示。

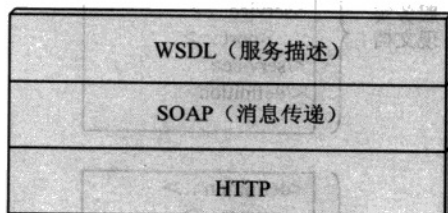


图 13.2 Web 服务协议栈

SOAP 是 Simple Object Access Protocol 的简称，翻译为“简单对象访问协议”。它是 XML Web Services 的通信协议，这种协议采用 XML 消息格式，在分布环境中可提供一种简单、轻量的交换结构化和类型信息的机制。实际上，SOAP 本身并没有定义任何应用程序语义，而是通过提供一个有标准组件的包模型和在模块中编码数据的方式来定义一个简单的表示应用程序语义的机制。在这种机制下，SOAP 就可以利用 HTTP 等开放的网络协议来进行消息传递。

SOAP 运行于 HTTP 之上，可以“穿透”防火墙，利用这种协议可以实现与平台无关的访问功能。特别是在 Internet 上，由于不同企业、不同用户所采用的操作系统设置了各种各样的防火墙，SOAP 的这种跨平台特性就显得尤为重要。

总的来说，SOAP 提供了跨平台访问服务、对象和服务的技术。通过利用 SOAP，就可以使用 Internet 上提供的服务、查询相关信息，而不用考虑这些服务在哪里、如何实现等。这就为有效地利用 Internet 上丰富的资源奠定了必要的基础。

13.2.2 WSDL

WSDL 是一种基于 XML 格式、用于描述 Web 服务的语言，称为 Web 服务描述语言。这

种语言定义了 Web 服务支持的消息，以及消息里面包含的组件。WSDL 文档可用于动态发布 Web 服务、查找已发布的 Web 服务及绑定 Web 服务。

Web 服务描述语言将 Web 服务定义成一系列的端口 (port)，每个端口用来表示从抽象端口类型 (port type) 到用于 Web 服务的具体通信协议的一个映射。端口类型由一组与 Service Provider 交换信息的操作组成，它支持对包含消息的数据类型的定义。

一个完整的 WSDL 服务描述是由一个服务接口文档和一个服务实现文档组成的。WSDL 服务实现文档包含 import 元素和 service 元素以及包含实现一个服务接口文档的服务描述。import 元素中至少会有一个包含对 WSDL 服务接口文档的引用 (当然也可以包含多个服务接口文档的引用)。

服务接口由 WSDL 文档来描述，该文档包含 types、import、message、portType 和 binding 等元素。服务接口是 Web 服务的抽象定义，并被用于描述某种特定类型的服务，它是用于实现一个或多个服务的 WSDL 服务定义。import 元素用于在一个服务接口文档中引用另一个服务接口文档。

服务实现和服务接口文档的结构及其关系如图 13.3 所示。

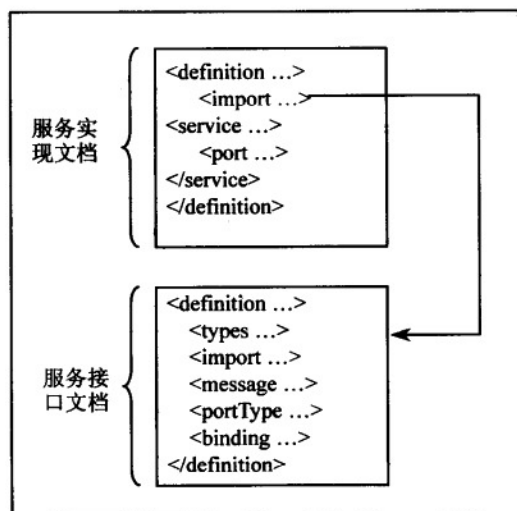


图 13.3 WSDL 文档结构

13.2.3 UDDI

UDDI (Universal Description, Discovery and Integration)，意为统一描述、发现和整合。它通过 Internet 来描述服务、发现业务，并且整合业务服务，以建立一个平台独立、开放的框架。UDDI 主要通过服务注册，以及使用 SOAP 访问这些注册信息的约定来实现上述目标。

UDDI 计划的核心组件是 UDDI 商业注册，它使用一个 XML 文档来描述企业及其提供的 Web 服务。UDDI 商业注册所提供的信息包含以下三个部分：

- 白页 (White Page)：包括了企业的名称、地址、联系方法等。
- 黄页 (Yellow Page)：包括了基于标准分类法的行业类别。

- 绿页 (Green Page): 包括了指向有关文件或 URL 的指针, 该指针指向该企业所提供的 Web 服务的技术信息, 以帮助用户编写应用程序、使用 Web 服务。

13.3 创建 Web 服务


Delphi 2005 提供了对 Web 服务开发的完美支持, 可以通过 ASP.NET 快速地使用 Delphi 2005 IDE 开发 Web 服务应用程序。

13.3.1 创建 Web 服务应用程序的基本步骤

前面介绍了 Web 服务的特点和体系结构, 知道它是基于多层协议之上的一种应用程序。但是在 Delphi 2005 中创建一个 Web 服务应用程序却是非常容易的, 不必了解复杂的协议。可按照下列介绍的步骤创建一个 Web 服务应用程序。

(1) 在 Delphi 2005 中, 选择菜单 File→New→Other... 命令, 将打开 New Items 对话框。在 New Items 对话框左边的方框中选择 Delphi for .NET Projects 结点, 在右边方框中选择 ASP.NET Web Service Application 图标, 然后单击 OK 按钮, 将出现 New ASP.NET Web Service Application 对话框。

(2) 在 New ASP.NET Web Service Application 对话框中设置 Web 服务程序的名称、存放位置及使用的 Web 服务器。在这个实例中程序名为 FirstWebServiceApp, 其他项的设置采用默认值。

(3) 单击 OK 按钮, ASP.NET Web 服务程序 FirstWebServiceApp 创建完毕。但现在这个程序还只是一个空框架, 没有提供任何功能。单击运行按钮 , 运行该程序, 结果如图 13.4 所示。

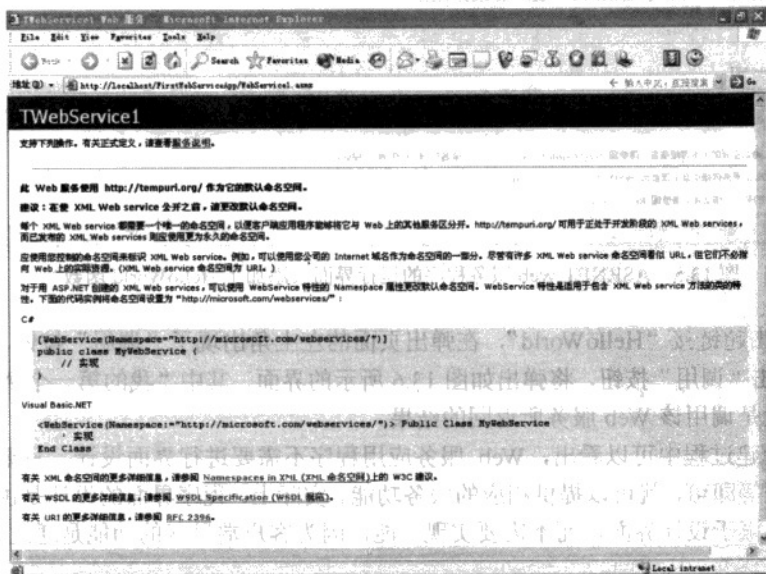



图 13.4 (空白的) ASP.NET Web 服务程序的运行界面

(4) 为使程序具有相应的功能，要放开程序自带的 HelloWorld 函数（默认是被注释了），方法是：在工程管理器中找到并双击 WebService1.pas 结点，打开 WebService1.pas 文件，去掉对函数 HelloWorld 的注释（有两个地方），并把语句“Result := 'Hello World;”改为“Result := '我的第一个 ASP.NET Web 服务程序;”，更改后 HelloWorld 函数代码如下：

```
function TWebService1.HelloWorld: string;
begin
    Result := '我的第一个 ASP.NET Web 服务程序';
end;
```

实际上，[WebMethod]就是表示该方法可以提供 Web 服务的方法，即在程序运行后，HelloWorld 函数将是 Web 服务的一个组成部分，可供其他应用程序调用。

(5) 单击运行按钮，运行修改后的程序，结果如图 13.5 所示。这时发现运行界面的左上角多了一个超链接“HelloWorld”。

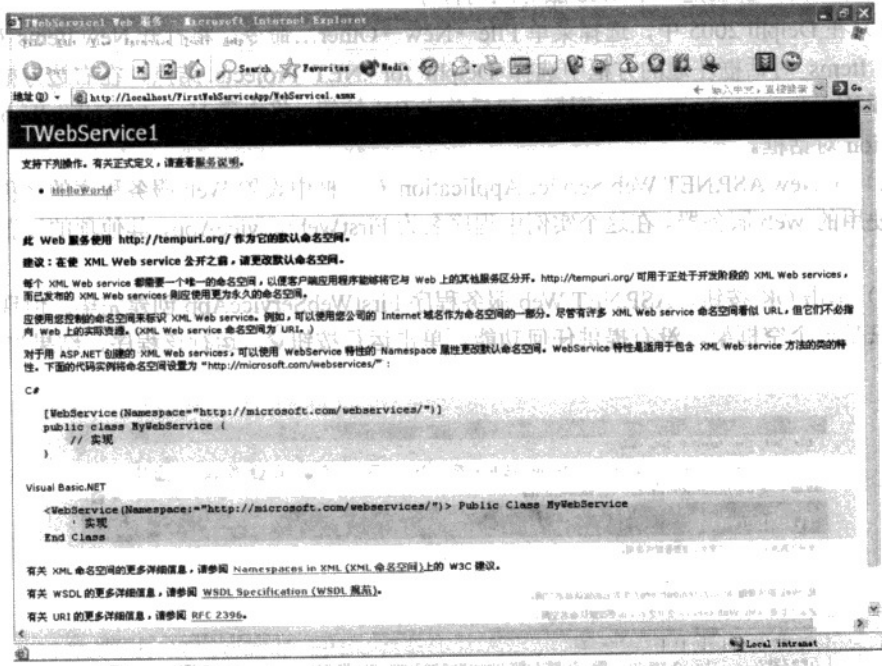


图 13.5 ASP.NET Web 服务程序的运行界面（添加了 HelloWorld 函数）

(6) 单击超链接“HelloWorld”，在弹出页面的左上角出现了“调用”按钮。

(7) 单击“调用”按钮，将弹出如图 13.6 所示的界面，其中“我的第一个 ASP.NET Web 服务程序”就是调用该 Web 服务所返回的结果。

从上述创建过程中可以看出，Web 服务应用程序不需要进行界面设计，只要编写好有关的函数和过程等即可，就可以提供相应的服务功能。实际上，程序界面的设计是由客户端来完成的，但也仅限于设计界面，而不需要实现功能，因为客户端呈现的功能是通过调用 Web 服务来实现的。

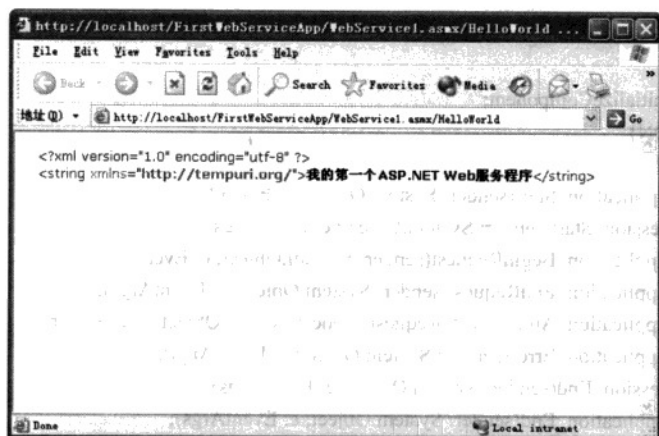


图 13.6 调用 Web 服务所返回的结果

13.3.2 熟悉 Web 服务程序的文件结构

打开上节创建的 Web 服务程序 FirstWebServiceApp 的工程管理器，如图 13.7 所示。



图 13.7 Web 服务程序 FirstWebServiceApp 的工程管理器

一个 Web 服务程序主要由以下文件组成。

1. Global.asax 文件及 Global.pas 文件

该文件与 ASP.NET 应用程序的 Global.asax 文件的功能相似，它主要用于处理一些应用程序级的事件。Global.asax 文件还包含了 Global.pas 文件。实际上，Global.asax 文件仅包含一行代码：

```
<%@ Application Codebehind="Global.pas" Inherits="Global.TGlobal" %>
```

该指令指明了 Global.asax 文件的代码是在 Global.pas 文件中。Global.pas 文件代码如下：

```
unit Global;
interface
```

```
uses
```

```
System.Collections, System.ComponentModel,
System.Web, System.Web.SessionState;
```

```
type
```

```
TGlobal = class(System.Web.HttpApplication)
```

```

{$REGION 'Designer Managed Code'}
strict private
    procedure InitializeComponent;
{$ENDREGION}
strict protected
    procedure Application_Start(sender: System.Object; e: EventArgs);
    procedure Session_Start(sender: System.Object; e: EventArgs);
    procedure Application_BeginRequest(sender: System.Object; e: EventArgs);
    procedure Application_EndRequest(sender: System.Object; e: EventArgs);
    procedure Application_AuthenticateRequest(sender: System.Object; e: EventArgs);
    procedure Application_Error(sender: System.Object; e: EventArgs);
    procedure Session_End(sender: System.Object; e: EventArgs);
    procedure Application_End(sender: System.Object; e: EventArgs);
private
    { Private Declarations }
public
    constructor Create;
end;

implementation
{$REGION 'Designer Managed Code'}
/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TGlobal.InitializeComponent;
begin
end;
{$ENDREGION}

constructor TGlobal.Create;
begin
    inherited;
    // Required for Designer support
    InitializeComponent;
    // TODO: Add any constructor code after InitializeComponent call
end;

procedure TGlobal.Application_Start(sender: System.Object; e: EventArgs);
begin
end;

procedure TGlobal.Session_Start(sender: System.Object; e: EventArgs);
begin
end;

```

```
procedure TGlobal.Application_BeginRequest(sender: System.Object; e: EventArgs);
begin
end;

procedure TGlobal.Application_EndRequest(sender: System.Object; e: EventArgs);
begin
end;

procedure TGlobal.Application_AuthenticateRequest(sender: System.Object; e: EventArgs);
begin
end;

procedure TGlobal.Application_Error(sender: System.Object; e: EventArgs);
begin
end;

procedure TGlobal.Session_End(sender: System.Object; e: EventArgs);
begin
end;

procedure TGlobal.Application_End(sender: System.Object; e: EventArgs);
begin
end;

end.
```

2. Web.config 文件

Web.config 文件用于设置 Web 服务程序的配置。例如，应用程序可以读取的设置；在页面文件中所采用默认的编程语言等。其代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <!-- DYNAMIC DEBUG COMPILATION
      Set compilation debug="true" to enable ASPX debugging. Otherwise, setting this value to
      false will improve runtime performance of this application.
      Set compilation debug="true" to insert debugging symbols (.pdb information)
      into the compiled page. Because this creates a larger file that executes
      more slowly, you should set this value to true only when debugging and to
      false at all other times. For more information, refer to the documentation about
      debugging ASP .NET files.
    -->

    <compilation
      defaultLanguage="c#"
      debug="true">
    <assemblies>
      <!-- BORLAND DEBUG KERNEL
```



```

    The httpModule Borland.DbkAsp.DbkConnModule is required for debugging under IIS.
    This module may be installed in web.config or machine.config, but not both.
    Move the following line outside this comment to install.
    <add assembly="Borland.dbkasp, Version=9.0.0.1, Culture=neutral,
        PublicKeyToken=b0524c541232aae7"/>
    -->
</assemblies>
</compilation>

<httpModules>
<!-- BORLAND DEBUG KERNEL
    The httpModule Borland.DbkAsp.DbkConnModule is required for debugging under IIS.
    This module may be installed in web.config or machine.config, but not both.
    Move the following line outside this comment to install.
    <add name="DbgConnect"
        type="Borland.DbkAsp.DbkConnModule,Borland.dbkasp,
            Version=9.0.0.1,Culture=neutral,PublicKeyToken=b0524c541232aae7"/>
    -->
</httpModules>

<!-- CUSTOM ERROR MESSAGES
    Set customError mode values to control the display of user-friendly
    error messages to users instead of error details (including a stack trace):

    "On" Always display custom (friendly) messages
    "Off" Always display detailed ASP.NET error information.
    "RemoteOnly" Display custom (friendly) messages only to users not running
    on the local Web server. This setting is recommended for security purposes, so
    that you do not display application detail information to remote clients.
    -->
<customErrors
    mode="RemoteOnly"
/>

<!-- AUTHENTICATION
    This section sets the authentication policies of the application.
    Possible modes are "Windows", "Forms", "Passport" and "None"
    -->
<authentication mode="Windows" />

<!-- APPLICATION-LEVEL TRACE LOGGING
    Application-level tracing enables trace log output for every page within an application.
    Set trace enabled="true" to enable application trace logging. If pageOutput="true", the
    trace information will be displayed at the bottom of each page. Otherwise, you can view the
    application trace log by browsing the "trace.axd" page from your web application
    root.

```

```

-->
<trace
    enabled="false"
    requestLimit="10"
    pageOutput="false"
    traceMode="SortByTime"
localOnly="true"
/>

<!-- SESSION STATE SETTINGS
    By default ASP .NET uses cookies to identify which requests belong to a particular session.
    If cookies are not available, a session can be tracked by adding a session identifier to
    the URL. To disable cookies, set sessionState cookieless="true".
-->
<sessionState
    mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user id=sa;password="
    cookieless="false"
    timeout="20"
/>

<!-- GLOBALIZATION
    This section sets the globalization settings of the application.
-->
<globalization
    requestEncoding="utf-8"
    responseEncoding="utf-8"
/>
</system.web>
</configuration>

```

3. .asmx 文件

.asmx 文件是页面文件，它仅包含一行代码：

```

<%@ WebService Language="c#" Debug="true" Codebehind="WebService1.pas"
    Class="WebService1.TWebService1" %>

```

该指令表明，页面文件的代码文件是 WebService1.pas 文件，页面文件的编程语言是 C#。

Web 服务要实现的功能（页面功能）主要是在这个文件中实现的，它是程序员的“用武之地”。

对于上节创建的 FirstWebServiceApp 程序，其 WebService1.pas 文件的代码如下：

```

unit WebService1;
interface
uses
    System.Collections, System.ComponentModel,
    System.Data, System.Diagnostics, System.Web,
    System.Web.Services;
type
    /// <summary>

```

```

/// Summary description for WebService1.
/// </summary>
TWebService1 = class(System.Web.Services.WebService)
{$REGION 'Designer Managed Code'}
strict private
    /// <summary>
    /// Required designer variable.
    /// </summary>
    components: IContainer;
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    procedure InitializeComponent;
{$ENDREGION}
strict protected
    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    procedure Dispose(disposing: boolean); override;
private
    { Private Declarations }
public
    constructor Create;
    // Sample Web Service Method
    [WebMethod]
    function HelloWorld: string;
    (**)
end;

implementation
{$REGION 'Designer Managed Code'}
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWebService1.InitializeComponent;
begin
end;
{$ENDREGION}

constructor TWebService1.Create;
begin
    inherited;
    //
    // Required for Designer support

```

```
//
InitializeComponent;
//
// TODO: Add any constructor code after InitializeComponent call
//
end;

/// <summary>
/// Clean up any resources being used.
/// </summary>
procedure TWebService1.Dispose(disposing: boolean);
begin
    if disposing and (components <> nil) then
        components.Dispose;
    inherited Dispose(disposing);
end;

// Sample Web Service Method
// The following method is provided to allow for testing a new web service.

function TWebService1.HelloWorld: string;
begin
    Result := '我的第一个 ASP.NET Web 服务程序';
end;
(**)

end.
```

在 WebService1.pas 文件中添加不同的代码将使 Web 服务程序提供更多的功能，本章下面的内容将介绍这方面的知识，并详细介绍如何使用这些 Web 服务。

13.4 Delphi 中 Web 服务应用的开发实例

作为一个例子，本节主要介绍如何开发一个具有一定应用价值的 Web 服务应用程序，通过这个程序的设计，读者可以掌握 Web 服务应用开发的一般方法。

待创建的 Web 服务可提供下面两个功能：

- 计算任意一个一元二次方程的根。
- 求任意一个圆的面积。

13.4.1 创建 Web 服务程序

可以按照下列方法创建这个 Web 服务程序。

在 Delphi 2005 中，选择菜单 File→New→Other...命令，将打开 New Items 对话框。在 New Items 对话框左边的方框中选择 Delphi for .NET Projects 结点，在右边方框中选择 ASP.NET Web Service Application 图标，然后单击 OK 按钮，将出现 New ASP.NET Web Service Application 对

话框。在此对话框中把待创建的程序名称设置为 SolutionOfEquationCircle, 然后单击 OK 按钮。至此, 一个名为 SolutionOfEquationCircle 的空白 Web 服务程序创建完毕, 详细的创建步骤说明可参见 13.3.1 节。

13.4.2 添加 Web 服务的功能

在工程管理器中找到并双击 WebService1.pas 结点, 在代码编辑器中打开 WebService1.pas 文件。然后在该文件中加入用于求解方程根的两个函数, 即 Equation_x1(a,b,c:double)和 Equation_x2(a,b,c:double)函数, 以及求圆面积的一个函数: Circle(r:double)函数。方法是: 首先在 public 部分声明这几个函数, 并在每一个函数的前一行加上[WebMethod], 结果 public 部分的代码如下:

```
public
    constructor Create;
    (*
    // Sample Web Service Method
    [WebMethod]
    function HelloWorld: string;
    *)
    [WebMethod]
    function Equation_x1(a,b,c:double): double;
    [WebMethod]
    function Equation_x2(a,b,c:double): double;
    [WebMethod]
    function Circle(r:double): double;
```

[WebMethod]是表示把紧接其后的函数设置为 Web 服务的一个组成部分, 在 Web 服务发布的时候它跟着被发布, 以供其他程序调用。

其次, 在 implementation 部分中对声明的函数和方法予以实现。为此, 在该部分加入下列代码:

```
function TWebService1.Equation_x1(a,b,c:double): double; //求方程的第一个根, a、b、c 为方程系数
var sqrtx:double;
begin
    sqrtx := Math.Sqrt(b*b-4*a*c);
    Result := (-b+sqrtx)/(2*a);
end;

function TWebService1.Equation_x2(a,b,c:double): double; //求方程的第二个根, a、b、c 为方程系数
var sqrtx:double;
begin
    sqrtx := Math.Sqrt(b*b-4*a*c);
    Result := (-b-sqrtx)/(2*a);
end;

function TWebService1.Circle(r:double): double; //求圆的面积, r 为圆半径
begin
    Result := 3.14*r*r;
end;
```

13.4.3 发布 Web 服务

发布本来是件复杂的事情,但 Delphi 2005 中它却变成了一件轻而易举的事情。只要代码编写完成以后,编译运行该程序一次,Delphi 2005 将自动通过 IIS 发布对应的 Web 服务,此后其他程序就可以对这些发布的服务进行调用了。

运行后,程序界面左上角将出现三个超链接,如图 13.8 所示,这表示 Web 服务已经成功发布。



图 13.8 已成功发布 Web 服务

如何调用一个已经发布了的 Web 服务,这是下一节要介绍的内容。

13.5 Web 服务的访问实例

访问 Web 服务可以在两种模式下实现:一种是在 Web 页面中实现,另一种是在 Windows 窗体中实现。以下分别讲述。

13.5.1 创建 Web 窗体客户端程序

(1) 在 Delphi 2005 IDE 中选择菜单 File→New→ASP.NET Web Application - Delphi for .NET 命令,打开 New ASP.NET Web Application 对话框,在此设置 Web 窗体客户端程序的名称——SolutionClient。

(2) 设置完成后单击 OK 按钮,进入 Web 窗体设计界面。

(3) 选择菜单 Project→Add Web Reference...命令,打开 Add Web Reference 对话框。该对话框用于设置 Web 服务的一个 WSDL 文件位置,在此输入发布页面的 URL 地址,即:

`http://localhost/SolutionOfEquationCircle/WebService1.aspx`

当然,该地址是由 Web 服务开发者提供的。然后单击该对话框右上角的 go 按钮,如果一切顺利,将出现如图 13.9 所示的界面。该结果表示成功找到了指定的 Web 服务。

(4) 单击右下角的 Add Reference 按钮,添加 Web 引用的操作全部完成。这时,工程管理器中将增加一个结点——Web Refenece,展开该结点。此后,在以后的编程中就可以引用这个 Web 服务了。

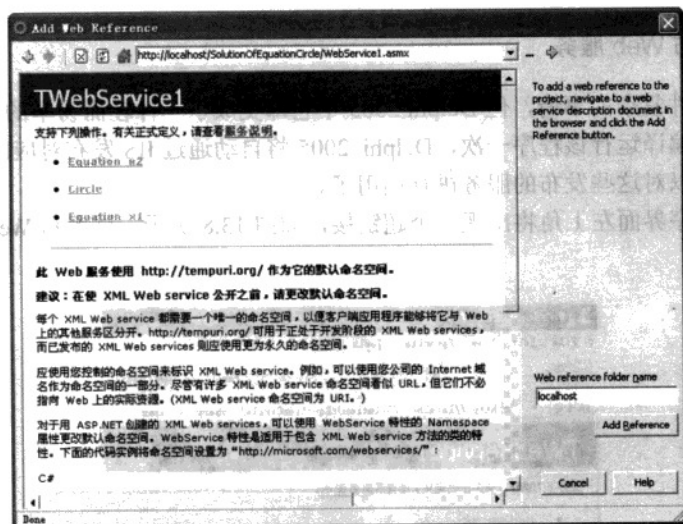


图 13.9 添加 Web 引用

(5) 设计 Web 窗体。此处的 Web 窗体程序属于客户端程序，它的基本功能不是计算，而是接收数据、向 Web 服务请求有关计算任务、最后把得到的结果显示出来。对于本例而言，它首先是要求用户输入方程的系数和圆的半径，然后请求服务，最后把方程的两个根及圆的面积显示出来。为此，可以把 Web 页面设计成如图 13.10 所示的界面。

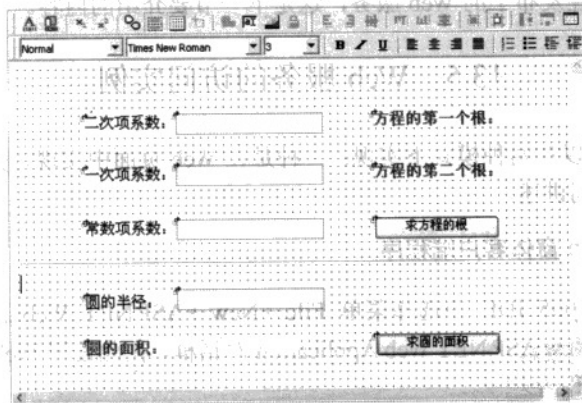


图 13.10 Web 窗体设计界面 (客户端)

其中，各组件属性 (Web Controls 标签下的组件) 的设置情况如表 13.1 所示。

表 13.1 各组件属性的设置

组件类别	组件名称 (ID 属性值)	属性设置项目	设置结果
Label	Label1	Text	二次项系数:
	Label2	Text	一次项系数:
	Label3	Text	常数项系数:

续表

组件类别	组件名称 (ID 属性值)	属性设置项目	设置结果
Label	Label4	Text	方程的第一个根:
		Font.Bold	true
		ForeColor	Red
	Label5	Text	方程的第二个根:
		Font.Bold	true
		ForeColor	Red
	Label6	Text	圆的半径:
	Label7	Text	圆的面积:
		Font.Bold	true
		ForeColor	Red
TextBox	TextBox1	Text	(空值)
	TextBox2	Text	(空值)
	TextBox3	Text	(空值)
	TextBox4	Text	(空值)
Button	Button1	Text	求方程的根
	Button2	Text	求圆的面积

(6) 代码设计。代码设计就是为各按钮添加事件处理代码。以下是“求方程的根”和“求圆的面积”这两个按钮的事件处理过程:

////// “求方程的根”按钮 (Button1) 的事件处理过程 ////

```
procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
```

```
var EquationSer:TWebService1;
```

```
result:double;
```

```
begin
```

```
EquationSer := TWebService1.Create;
```

```
//调用 Web 服务, 求方程的第一个根 x1
```

```
result := EquationSer.Equation_x1(StrToFloat(TextBox1.Text ), StrToFloat(TextBox2.Text ), StrToFloat(TextBox3.Text));
```

```
label4.Text:= '方程的第一个根: x1= '+FloatToStr(result);
```

```
//调用 Web 服务, 求方程的第二个根 x2
```

```
result := EquationSer.Equation_x2(StrToFloat(TextBox1.Text), StrToFloat(TextBox2.Text), StrToFloat(TextBox3.Text));
```

```
label5.Text:= '方程的第二个根: x2= '+FloatToStr(result);
```

```
end;
```

////// “求圆的面积”按钮 (Button2) 的事件处理过程 ////

```
procedure TWebForm1.Button2_Click(sender: System.Object; e: System.EventArgs);
```

```
var CircleSer:TWebService1;
```

```
result:double;
```


begin

```
CircleSer := TWebService1.Create;
//调用 Web 服务, 求圆的面积
result := CircleSer.Circle(StrToFloat(TextBox4.Text));
label7.Text := '圆的面积: S = '+FloatToStr(result);
```

end;

要使用数据类型转换函数 FloatToStr, 必须在 uses 部分加入 SysUtils (因为默认情况不包含该单元)。

这两个过程都使用了类 TWebService1 来声明对象, 所以在程序中必须引入该类。为此, 选择菜单 File→Use Unit ...命令, 将打开 Use Unit 对话框; 在该对话框中选择 localhost1.WebService1 选项, 如图 13.11 所示, 然后单击 OK 按钮即可(因为该类已在 localhost1.WebService1.pas 文件中定义, 可从工程管理器查看)。

上述两个过程在 WebForm1.pas 文件中, 该文件完整的代码如下:

```
unit WebForm1;
interface

uses
  System.Collections, System.ComponentModel, SysUtils,
  System.Data, System.Drawing, System.Web, System.Web.SessionState,
  System.Web.UI, System.Web.UI.WebControls, System.Web.UI.HtmlControls;

type
  TWebForm1 = class(System.Web.UI.Page)
  {$REGION 'Designer Managed Code'}
  strict private
    procedure InitializeComponent;
    procedure Button1_Click(sender: System.Object; e: System.EventArgs);
    procedure Button2_Click(sender: System.Object; e: System.EventArgs);
  {$ENDREGION}
  strict private
    procedure Page_Load(sender: System.Object; e: System.EventArgs);
  strict protected
    Label1: System.Web.UI.WebControls.Label;
    Label2: System.Web.UI.WebControls.Label;
    Label3: System.Web.UI.WebControls.Label;
    TextBox1: System.Web.UI.WebControls.TextBox;
    TextBox2: System.Web.UI.WebControls.TextBox;
    TextBox3: System.Web.UI.WebControls.TextBox;
    Label4: System.Web.UI.WebControls.Label;
    Label5: System.Web.UI.WebControls.Label;
    Button1: System.Web.UI.WebControls.Button;
```

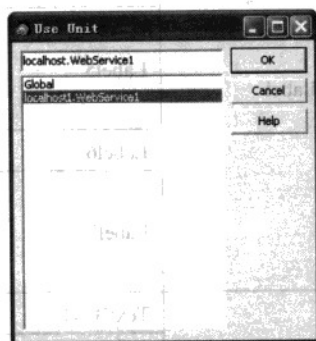


图 13.11 Use Unit 对话框

```

Label6: System.Web.UI.WebControls.Label;
TextBox4: System.Web.UI.WebControls.TextBox;
Label7: System.Web.UI.WebControls.Label;
Button2: System.Web.UI.WebControls.Button;
procedure OnInit(e: EventArgs); override;
private
    { Private Declarations }
public
    { Public Declarations }
end;

implementation
uses localhost1.WebService1;
{$REGION 'Designer Managed Code'}
/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWebForm1.InitializeComponent;
begin
    Include(Self.Button1.Click, Self.Button1_Click);
    Include(Self.Button2.Click, Self.Button2_Click);
    Include(Self.Load, Self.Page_Load);
end;
{$ENDREGION}

procedure TWebForm1.Page_Load(sender: System.Object; e: System.EventArgs);
begin
    // TODO: Put user code to initialize the page here
end;

procedure TWebForm1.OnInit(e: EventArgs);
begin
    // Required for Designer support
    InitializeComponent;
    inherited OnInit(e);
end;

procedure TWebForm1.Button1_Click(sender: System.Object; e: System.EventArgs);
var EquationSer: TWebService1;
    result: double;
begin
    EquationSer := TWebService1.Create;
    //调用 Web 服务, 求方程的第一个根 x1
    result := EquationSer.Equation_x1(StrToFloat(TextBox1.Text), StrToFloat(TextBox2.Text),
    StrToFloat(TextBox3.Text));

```

```

label4.Text:='方程的第一个根: x1= '+FloatToStr(result);
//调用 Web 服务, 求方程的第二个根 x2
result := EquationSer.Equation_x2(StrToFloat(TextBox1.Text), StrToFloat(TextBox2.Text),
StrToFloat(TextBox3.Text));
label5.Text:='方程的第二个根: x2= '+FloatToStr(result);
end;

```

```

procedure TWebForm1.Button2_Click(sender: System.Object; e: System.EventArgs);
var CircleSer:TWebService1;
    result:double;
begin
    CircleSer := TWebService1.Create;
    //调用 Web 服务, 求圆的面积
    result := CircleSer.Circle(StrToFloat(TextBox4.Text));
    label7.Text:='圆的面积: S = '+FloatToStr(result);
end;

```

以下是 WebForm1.aspx 页面文件的代码（它是在添加 Web Controls 组件等元素后自动产生的）:

```

<form runat="server">
  <p>
    <asp:Label id="Label1" style="Z-INDEX: 1; LEFT: 78px; POSITION: absolute; TOP: 56px"
      runat="server" height="19px" width="123px">二次项系数: </asp:Label>
    <asp:Label id="Label2"
      style="Z-INDEX: 1; LEFT: 78px; POSITION: absolute; TOP: 111px"
      runat="server" height="19px" width="123px">一次项系数: </asp:Label>
    <asp:Label id="Label3"
      style="Z-INDEX: 1; LEFT: 78px; POSITION: absolute; TOP: 168px"
      runat="server" height="19px" width="123px">常数项系数: </asp:Label>
    <asp:TextBox id="TextBox1"
      style="Z-INDEX: 2; LEFT: 174px; POSITION: absolute; TOP: 54px"
      runat="server"></asp:TextBox>
    <asp:TextBox id="TextBox2"
      style="Z-INDEX: 2; LEFT: 174px; POSITION: absolute; TOP: 109px"
      runat="server"></asp:TextBox>
    <asp:TextBox id="TextBox3"
      style="Z-INDEX: 2; LEFT: 174px; POSITION: absolute; TOP: 166px"
      runat="server"></asp:TextBox>
    <asp:Label id="Label4"
      style="Z-INDEX: 3; LEFT: 382px; POSITION: absolute; TOP: 54px"
      runat="server" width="219px" forecolor="Red" font-bold="True">方程的第一个根:
  </asp:Label>
    <asp:Label id="Label5"
      style="Z-INDEX: 3; LEFT: 382px; POSITION: absolute; TOP: 110px"
      runat="server" width="219px" forecolor="Red" font-bold="True">方程的第二个根:
  </asp:Label>
    <asp:Button id="Button1"

```

```

style="Z-INDEX: 4; LEFT: 382px; POSITION: absolute; TOP: 166px"
runat="server" width="131px" text="求方程的根"></asp:Button>
</p>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>&nbsp;</p>
<p>
    <hr>
</p>
<p></p>
<asp:Label id="Label6" style="Z-INDEX: 5; LEFT: 78px; POSITION: absolute; TOP: 246px"
runat="server" width="93px">圆的半径: </asp:Label>
<asp:TextBox id="TextBox4"
style="Z-INDEX: 6; LEFT: 174px; POSITION: absolute; TOP: 240px"
runat="server"></asp:TextBox>
<asp:Label id="Label7" style="Z-INDEX: 3; LEFT: 78px; POSITION: absolute; TOP: 294px"
runat="server" width="219px" forecolor="Red" font-bold="True">圆的面积: </asp:Label>
<asp:Button id="Button2"
style="Z-INDEX: 7; LEFT: 382px; POSITION: absolute; TOP: 289px"
runat="server" width="131px" text="求圆的面积"></asp:Button>
</form>

```

(7) 编译和运行 Web 窗体客户端程序了。按照上述方法编写代码后, 就可以运行该程序了。在运行界面 (Web 页面) 中输入方程的各项系数及圆的半径, 然后分别单击“求方程的根”和“求圆的面积”按钮, 即可显示相应的结果。图 13.12 表示了求方程“ $x^2+5x+6=0$ ”的根以及求半径为 4 的圆面积的结果。

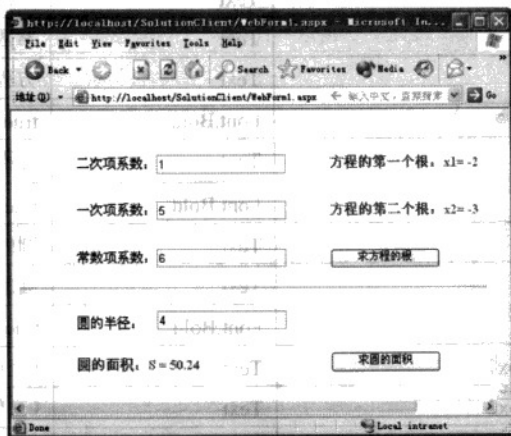


图 13.12 客户端程序的运行结果 (Web 界面)

13.5.2 创建 Windows 客户端程序

(1) 在 Delphi 2005 IDE 中选择菜单 File→New→Windows Forms Application-Delphi

for .NET 命令，创建 Windows 应用程序，并把该程序保存为 WinSolutionClient.dpr 文件名。

(2) 在窗体中加入 GroupBox 组件两个、Label 组件七个、TextBox 组件四个、Button 组件两个，并适当调整各组件的位置和大小，如图 13.13 所示；同时设置各组件的属性，如表 13.2 所示。

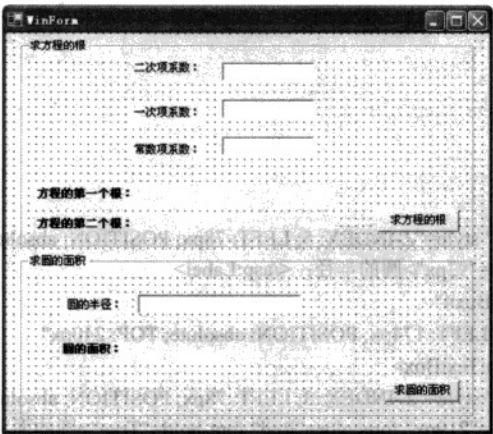



图 13.13 Windows 客户端程序的设计界面

表 13.2 各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
GroupBox	GroupBox1	Text	求方程的根
	GroupBox2	Text	求圆的面积
Label	Label1	Text	二次项系数:
	Label2	Text	一次项系数:
	Label3	Text	常数项系数:
	Label4	Text	方程的第一个根:
		Font.Bold	true
	Label5	Text	方程的第二个根:
		Font.Bold	true
TextBox	Label6	Text	圆的半径:
	Label7	Text	圆的面积:
		Font.Bold	true
	TextBox1	Text	(空值)
	TextBox2	Text	(空值)
	TextBox3	Text	(空值)
	TextBox4	Text	(空值)
Button	Button1	Text	求方程的根
	Button2	Text	求圆的面积

(3) 选择菜单 Project→Add Web Reference...命令, 打开 Add Web Reference 对话框。在该对话框的文本框中输入下列 URL 地址:

```
http://localhost/SolutionOfEquationCircle/WebService1.asmx
```

然后单击该对话框右上角的 go 按钮 , 最后单击右下角的 Add Reference 按钮, 添加 Web 引用的操作全部完成。这时, 工程管理器中将增加一个结点——Web Refenece。在以后的编程中就可以直接引用这个 Web 服务了。

(4) 代码设计。主要是对两个按钮编写事件处理代码, 当然也要把用到的 TWebService1 类引入程序, 并在 uses 部分加上 SysUtils (具体操作方法见上节), 最后得到的 WinForm.pas 文件的代码如下:

```
unit WinForm;
interface
uses
    System.Drawing, System.Collections, System.ComponentModel,
    System.Windows.Forms, System.Data, SysUtils;

type
    TWinForm = class(System.Windows.Forms.Form)
    {$REGION 'Designer Managed Code'}
    strict private
        /// <summary>
        /// Required designer variable.
        /// </summary>
        Components: System.ComponentModel.Container;
        Button1: System.Windows.Forms.Button;
        TextBox1: System.Windows.Forms.TextBox;
        TextBox2: System.Windows.Forms.TextBox;
        TextBox3: System.Windows.Forms.TextBox;
        Label1: System.Windows.Forms.Label;
        GroupBox1: System.Windows.Forms.GroupBox;
        Label2: System.Windows.Forms.Label;
        Label3: System.Windows.Forms.Label;
        Label4: System.Windows.Forms.Label;
        Label5: System.Windows.Forms.Label;
        GroupBox2: System.Windows.Forms.GroupBox;
        Label6: System.Windows.Forms.Label;
        TextBox4: System.Windows.Forms.TextBox;
        Button2: System.Windows.Forms.Button;
        Label7: System.Windows.Forms.Label;
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        procedure InitializeComponent;
        procedure Button1_Click(sender: System.Object; e: System.EventArgs);
        procedure Button2_Click(sender: System.Object; e: System.EventArgs);
```

```

{$ENDREGION}
strict protected
  /// <summary>
  /// Clean up any resources being used.
  /// </summary>
  procedure Dispose(Disposing: Boolean); override;
private
  { Private Declarations }
public
  constructor Create;
end;

[assembly: RuntimeRequiredAttribute(TypeOf(TWinForm))]
implementation
uses localhost.WebService1;
{$AUTOBOX ON}
{$REGION 'Windows Form Designer generated code'}

/// <summary>
/// Required method for Designer support -- do not modify
/// the contents of this method with the code editor.
/// </summary>
procedure TWinForm.InitializeComponent;
begin
  Self.Button1 := System.Windows.Forms.Button.Create;
  Self.TextBox1 := System.Windows.Forms.TextBox.Create;
  Self.TextBox2 := System.Windows.Forms.TextBox.Create;
  Self.TextBox3 := System.Windows.Forms.TextBox.Create;
  Self.Label1 := System.Windows.Forms.Label.Create;
  Self.GroupBox1 := System.Windows.Forms.GroupBox.Create;
  Self.Label5 := System.Windows.Forms.Label.Create;
  Self.Label4 := System.Windows.Forms.Label.Create;
  Self.Label3 := System.Windows.Forms.Label.Create;
  Self.Label2 := System.Windows.Forms.Label.Create;
  Self.GroupBox2 := System.Windows.Forms.GroupBox.Create;
  Self.Label7 := System.Windows.Forms.Label.Create;
  Self.Button2 := System.Windows.Forms.Button.Create;
  Self.TextBox4 := System.Windows.Forms.TextBox.Create;
  Self.Label6 := System.Windows.Forms.Label.Create;
  Self.GroupBox1.SuspendLayout;
  Self.GroupBox2.SuspendLayout;
  Self.SuspendLayout;
  //
  // Button1
  //
  Self.Button1.Location := System.Drawing.Point.Create(384, 184);

```

```
Self.Button1.Name := 'Button1';
Self.Button1.Size := System.Drawing.Size.Create(88, 23);
Self.Button1.TabIndex := 0;
Self.Button1.Text := '求方程的根';
Include(Self.Button1.Click, Self.Button1_Click);
//
// TextBox1
//
Self.TextBox1.Location := System.Drawing.Point.Create(216, 24);
Self.TextBox1.Name := 'TextBox1';
Self.TextBox1.TabIndex := 1;
Self.TextBox1.Text := "";
//
// TextBox2
//
Self.TextBox2.Location := System.Drawing.Point.Create(216, 64);
Self.TextBox2.Name := 'TextBox2';
Self.TextBox2.TabIndex := 2;
Self.TextBox2.Text := "";
//
// TextBox3
//
Self.TextBox3.Location := System.Drawing.Point.Create(216, 104);
Self.TextBox3.Name := 'TextBox3';
Self.TextBox3.TabIndex := 3;
Self.TextBox3.Text := "";
//
// Label1
//
Self.Label1.Location := System.Drawing.Point.Create(120, 24);
Self.Label1.Name := 'Label1';
Self.Label1.TabIndex := 4;
Self.Label1.Text := '二次项系数: ';
//
// GroupBox1
//
Self.GroupBox1.Controls.Add(Self.Label5);
Self.GroupBox1.Controls.Add(Self.Label4);
Self.GroupBox1.Controls.Add(Self.Label3);
Self.GroupBox1.Controls.Add(Self.Label2);
Self.GroupBox1.Controls.Add(Self.TextBox1);
Self.GroupBox1.Controls.Add(Self.TextBox2);
Self.GroupBox1.Controls.Add(Self.TextBox3);
Self.GroupBox1.Controls.Add(Self.Label1);
Self.GroupBox1.Controls.Add(Self.Button1);
Self.GroupBox1.Location := System.Drawing.Point.Create(16, 8);
```



```
Self.GroupBox1.Name := 'GroupBox1';
Self.GroupBox1.Size := System.Drawing.Size.Create(488, 224);
Self.GroupBox1.TabIndex := 5;
Self.GroupBox1.TabStop := False;
Self.GroupBox1.Text := '求方程的根';
//
// Label5
//
Self.Label5.Font := System.Drawing.Font.Create('宋体', 9, System.Drawing.FontStyle.Bold,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label5.Location := System.Drawing.Point.Create(16, 192);
Self.Label5.Name := 'Label5';
Self.Label5.Size := System.Drawing.Size.Create(328, 23);
Self.Label5.TabIndex := 8;
Self.Label5.Text := '方程的第二个根: ';
//
// Label4
//
Self.Label4.Font := System.Drawing.Font.Create('宋体', 9, System.Drawing.FontStyle.Bold,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label4.Location := System.Drawing.Point.Create(16, 160);
Self.Label4.Name := 'Label4';
Self.Label4.Size := System.Drawing.Size.Create(328, 23);
Self.Label4.TabIndex := 7;
Self.Label4.Text := '方程的第一个根: ';
//
// Label3
//
Self.Label3.Location := System.Drawing.Point.Create(120, 112);
Self.Label3.Name := 'Label3';
Self.Label3.Size := System.Drawing.Size.Create(80, 23);
Self.Label3.TabIndex := 6;
Self.Label3.Text := '常数项系数: ';
//
// Label2
//
Self.Label2.Location := System.Drawing.Point.Create(120, 72);
Self.Label2.Name := 'Label2';
Self.Label2.Size := System.Drawing.Size.Create(80, 23);
Self.Label2.TabIndex := 5;
Self.Label2.Text := '一次项系数: ';
//
// GroupBox2
//
Self.GroupBox2.Controls.Add(Self.Label7);
Self.GroupBox2.Controls.Add(Self.Button2);
```

```
Self.GroupBox2.Controls.Add(Self.TextBox4);
Self.GroupBox2.Controls.Add(Self.Label6);
Self.GroupBox2.Location := System.Drawing.Point.Create(16, 240);
Self.GroupBox2.Name := 'GroupBox2';
Self.GroupBox2.Size := System.Drawing.Size.Create(488, 176);
Self.GroupBox2.TabIndex := 6;
Self.GroupBox2.TabStop := False;
Self.GroupBox2.Text := '求圆的面积';
//
// Label7
//
Self.Label7.Font := System.Drawing.Font.Create('宋体', 9, System.Drawing.FontStyle.Bold,
    System.Drawing.GraphicsUnit.Point, (Byte(134)));
Self.Label7.Location := System.Drawing.Point.Create(43, 96);
Self.Label7.Name := 'Label7';
Self.Label7.Size := System.Drawing.Size.Create(317, 23);
Self.Label7.TabIndex := 3;
Self.Label7.Text := '圆的面积：';
//
// Button2
//
Self.Button2.Location := System.Drawing.Point.Create(392, 136);
Self.Button2.Name := 'Button2';
Self.Button2.Size := System.Drawing.Size.Create(80, 23);
Self.Button2.TabIndex := 2;
Self.Button2.Text := '求圆的面积';
Include(Self.Button2.Click, Self.Button2_Click);
//
// TextBox4
//
Self.TextBox4.Location := System.Drawing.Point.Create(126, 43);
Self.TextBox4.Name := 'TextBox4';
Self.TextBox4.Size := System.Drawing.Size.Create(176, 21);
Self.TextBox4.TabIndex := 1;
Self.TextBox4.Text := '';
//
// Label6
//
Self.Label6.Location := System.Drawing.Point.Create(48, 48);
Self.Label6.Name := 'Label6';
Self.Label6.Size := System.Drawing.Size.Create(72, 23);
Self.Label6.TabIndex := 0;
Self.Label6.Text := '圆的半径：';
//
// TWinForm
//
```

```

Self.AutoScaleBaseSize := System.Drawing.Size.Create(6, 14);
Self.ClientSize := System.Drawing.Size.Create(520, 430);
Self.Controls.Add(Self.GroupBox2);
Self.Controls.Add(Self.GroupBox1);
Self.Name := 'TWinForm';
Self.Text := 'WinForm';
Self.GroupBox1.ResumeLayout(False);
Self.GroupBox2.ResumeLayout(False);
Self.ResumeLayout(False);
end;
{$ENDREGION}

procedure TWinForm.Dispose(Disposing: Boolean);
begin
    if Disposing then
    begin
        if Components <> nil then
            Components.Dispose();
        end;
        inherited Dispose(Disposing);
    end;
end;

constructor TWinForm.Create;
begin
    inherited Create;
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent;
    //
    // TODO: Add any constructor code after InitializeComponent call
    //
end;

procedure TWinForm.Button2_Click(sender: System.Object; e: System.EventArgs);
var CircleSer: TWebService1;
    result: double;
begin
    CircleSer := TWebService1.Create;
    result := CircleSer.Circle(StrToFloat(TextBox4.Text));
    label7.Text := '圆的面积: S = '+FloatToStr(result);
end;

procedure TWinForm.Button1_Click(sender: System.Object; e: System.EventArgs);
var EquationSer: TWebService1;
    result: double;

```

```
begin
    EquationSer := TWebService1.Create;
    result := EquationSer.Equation_x1(StrToFloat(TextBox1.Text),
    StrToFloat(TextBox2.Text),
    StrToFloat(TextBox3.Text));
    label4.Text:='方程的第一个根: x1='+FloatToStr(result);
    result := EquationSer.Equation_x2(StrToFloat(TextBox1.Text),
    StrToFloat(TextBox2.Text),
    StrToFloat(TextBox3.Text));
    label5.Text:='方程的第二个根: x2='+FloatToStr(result);
end;
end.
```

(5) 编译和运行 Web 窗体客户端程序。代码编写完后运行该程序，在运行界面（Windows 界面）中输入方程的各项系数及圆的半径，然后分别单击“求方程的根”和“求圆的面积”按钮，即可显示相应的结果。图 13.14 表示了求方程“ $x^2+8x+4=0$ ”的根及求半径为 4.8 的圆面积的结果。

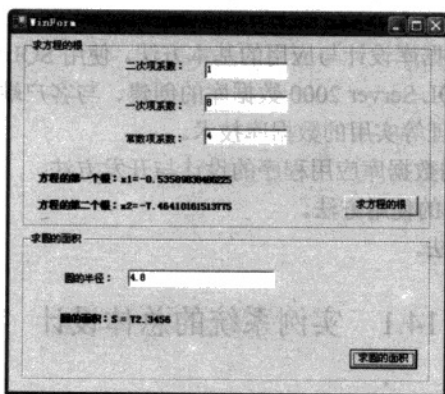


图 13.14 客户端程序的运行结果（Windows 界面）

由以上不难看出，编写 Web 界面客户端程序和编写 Windows 界面程序所用的代码几乎是一样的，但界面的风格却完全不同。可见，Delphi 2005 作为一种开发平台，其继承性和兼容性是非常好的，用户完全可以根据需要选择相应的风格或交叉使用这两种风格。

13.6 小结

本章介绍了 ASP.NET Web Services 应用程序的开发技术，涉及内容包括 Web 服务体系结构、Web 服务协议栈，以及 Web 服务应用的开发、Web 服务的访问等。通过对本章的学习，读者应掌握下列内容：

- Web 服务的特点，包括优缺点、体系结构和协议栈等。
- 创建 Web 服务应用程序的基本步骤及 Web 服务程序的文件结构。
- Web 服务应用的开发和访问技术，及开发的一般方法。

第 14 章 Delphi 2005 数据库应用开发实例

——学生信息管理系统

信息管理系统是以计算机为基础，由人和计算机结合的对信息进行收集、存储、维护、加工、传递和使用的一种管理系统，其目的是使人流、物流、资金流和信息流处于最佳状态，以最少的资源投入获得最佳的综合效益。学生信息管理系统主要包括学生信息管理模块、教师信息管理模块、课程信息管理模块、成绩信息管理模块和系统维护模块等部分。在校务管理中，为有关部门提供完整、综合、共享的信息，对于学校的教育管理、教学和科研等都有很大的实用价值。本章主要介绍利用 Delphi 2005 对学生信息管理系统进行设计与开发的过程，涉及要点如下：

- SQL Server 2000 数据库设计与应用的基本方法。使用 SQL Server 2000 作为后台的数据库系统，讲述 SQL Server 2000 数据库的创建、与客户端连接的方法，以及如何实现数据库的事务处理等实用的数据库技术。
- 基于 Delphi 2005 的数据库应用程序的设计与开发方法。
- Rave reports 报表器的使用方法。
- 安装程序的制作方法。

14.1 实例系统的总体设计

14.1.1 学生信息管理系统概述

在各大高校，对于学生的管理是相当重要的，而且也是相当麻烦的，它是学校管理中最基本的一项常规性工作。而长期以来，学校管理都是依赖人工来进行的，面对如此众多的学生信息，其工作量是相当巨大的。这样不仅浪费了大量的人力和物力，而且由于人工管理存在着大量的不可预知性，造成学生信息管理的一些不规范。

本实例是实现某个国际教育学院的学生信息管理系统，该系统对学生的基本信息和成绩进行管理，主要包括对学生的基本信息、教师信息、课程信息和成绩信息进行查询、修改、打印等基本功能。

根据学生信息管理系统的基本处理流程和实际要求，本系统需要实现以下功能：

- 学生基本信息的查询与修改。
- 对教师信息进行灵活的查询。
- 能够对学院基本课程信息进行查询和修改。
- 要求能够对学生成绩信息进行查询和修改，并能打印学生成绩单。

14.1.2 实例系统的功能描述

1. 学生信息管理模块

可以通过此模块来管理学生的信息, 学生基本的信息包括学号、姓名、性别、出生日期、专业等。具体的功能包括对学生信息的添加、修改、删除和查询等操作。为了方便管理, 其中提供根据专业、班级、学号和姓名等不同方式的查询。

2. 教师信息管理模块

通过此模块来管理教师的信息, 教师基本的信息包括教师编号、姓名、性别、出生日期、专业、职称、身份证号码、电话和住址等。具体的功能包括对教师信息的添加、修改、删除、查询等操作, 提供根据专业、教师编号、姓名等不同方式的查询。

3. 课程信息管理模块

通过此模块来管理课程信息, 课程相关基本信息包括课程编号、课程名称、学时、学分、选用课本、开出专业和开课时间等。具体的功能包括对课程信息的添加、修改、删除和查询等操作, 提供根据专业、课程编号、课程名称等不同方式的查询。

4. 成绩信息管理模块

通过此模块来管理学生的成绩信息, 学生成绩信息包括学号、姓名、课程名称、任课教师、平时成绩、期中成绩、期末成绩和期评成绩等。具体的功能包括对成绩信息的添加、修改、删除、查询和打印成绩单等操作, 提供根据班级、学号和姓名等不同方式的查询。

5. 系统维护模块

此模块主要是实现增加用户、删除用户、修改密码和分配权限等功能。

14.2 实例系统的数据库设计

根据该系统的需求, 数据库采用 Microsoft SQL Server 2000, 它完全适合本系统的工作需求, 它所支持的字段数据类型包括数值、字符、日期时间及二进制和图片等类型, 是当前的主流数据库产品之一。它还具有性能稳定、便于操作维护和具有较好的安全可靠等特点, 是作为服务器端数据库平台的理想选择。根据系统的要求, 系统用到的各种数据表如表 14.1 至表 14.6 所示。

表 14.1 学生信息表 stutable

名称	数据类型	长度	是否为空	说明
ID	int	4	NOT NULL	编号
studentno	char	10	NOT NULL	学号
name1	char	10	NULL	中文名
name2	char	30	NULL	英文名
sex	char	2	NULL	性别
birthday	datetime	8	NULL	出生日期
country	char	20	NULL	国籍
type	char	30	NULL	专业

续表

名称	数据类型	长度	是否为空	说明
dorm	char	30	NULL	宿舍
tell	char	12	NULL	宿舍电话
tel2	char	15	NULL	手提电话
intime	datetime	8	NULL	入学时间
outtime	datetime	8	NULL	毕业时间
destination	char	50	NULL	毕业去向
classno	char	10	NULL	班级号
remark	char	50	NULL	备注

表 14.2 教师信息表 teatable

名称	数据类型	长度	是否为空	说明
ID	int	4	NOT NULL	编号
teacherno	char	10	NOT NULL	教师编号
name	char	10	NULL	姓名
sex	char	8	NULL	性别
idcard	char	18	NULL	身份证号
tell	char	12	NULL	家庭电话
tel2	char	15	NULL	手提电话
email	varchar(50)	50	NULL	电子邮箱
address	char	50	NULL	住址
zhicheng	char	10	NULL	职称
xueli	char	10	NULL	学历
xuewei	char	10	NULL	学位
remark	char	50	NULL	备注

表 14.3 班级信息表 clatable

名称	数据类型	长度	是否为空	说明
ID	int	4	NOT NULL	编号
classno	char	10	NOT NULL	班级号
name	char	20	NULL	班级名称
type	datetime	8	NULL	专业
teacherno	char	10	NULL	班主任教师编号
intime	datetime	8	NULL	入学时间
remark	char	50	NULL	备注

表 14.4 课程信息表 ctable

名称	数据类型	长度	是否为空	说明
ID	int	4	NOT NULL	编号
courseno	char	10	NOT NULL	课程号
name	char	30	NULL	课程名
type	char	30	NULL	专业
xueshi	int	4	NULL	学时
xuefen	int	4	NULL	学分
time1	char	20	NULL	开课时间
book	char	50	NULL	选用教材
teacherno	char	10	NULL	教师编号
teachername	char	10	NULL	教师姓名
remark	char	50	NULL	备注

表 14.5 成绩信息表 restable

名称	数据类型	长度	是否为空	说明
ID	int	4	NOT NULL	编号
studentno	char	10	NOT NULL	学号
studentname	char	10	NOT NULL	姓名
classname	char	10	NULL	班级
courseno	char	10	NULL	课程号
coursename	char	10	NULL	课程名称
coursexueshi	int	4	NULL	课程学时
time1	char	10	NULL	开课时间
pingshi1	float	8	NULL	平时成绩
pingshi2	float	8	NULL	期中成绩
qimo	float	8	NULL	期末成绩
zongping	float	8	NULL	总评成绩
remark	char	50	NULL	备注

表 14.6 管理信息表 admtable

名称	数据类型	长度	是否为空	说明
ID	int	4	NOT NULL	编号
loginno	char	10	NOT NULL	登录号
username1	char	10	NULL	用户名
password1	char	10	NULL	密码
quanxian1	bit	1	NULL	系统权限
quanxian2	bit	1	NULL	一般权限
remark	char	50	NULL	备注

为了实现该系统各模块的功能,以上六个表中的数据存在着一些关联,它们之间的关联情况如图 14.1 所示。

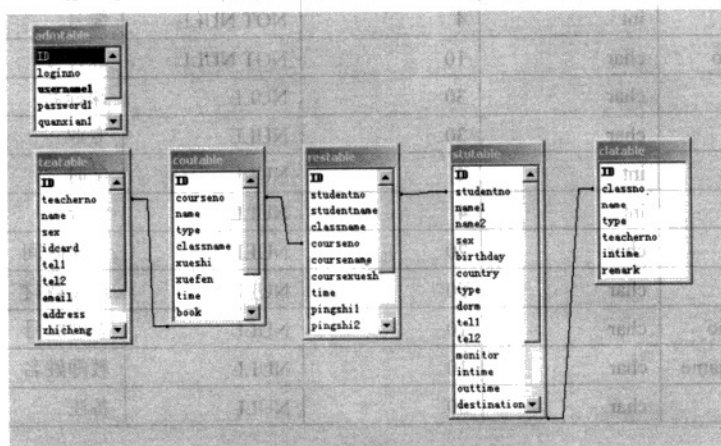


图 14.1 数据表之间的关联

14.3 创建实例系统的数据表

这一节主要向读者介绍如何在 SQL Server 2000 中创建以上所介绍的六个数据表。

14.3.1 创建新的数据库

在下面的步骤中,将在 SQL Server 2000 中建立一个名称为 xuesheng 的数据库。创建步骤如下:

(1) 首先在树状目录窗口中选取要建立新数据库的 SQL 服务器(YFSERVER),如图 14.2 所示。

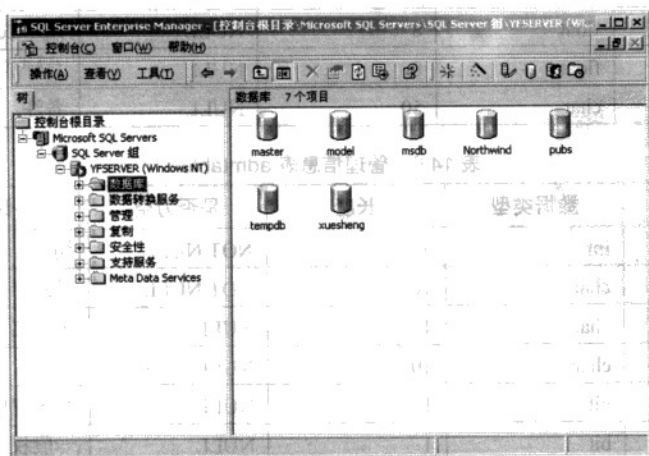



图 14.2 数据库画面

然后在工具栏上单击新增数据库工具按钮, 这时会打开如图 14.3 所示的“数据库属性”对话框。

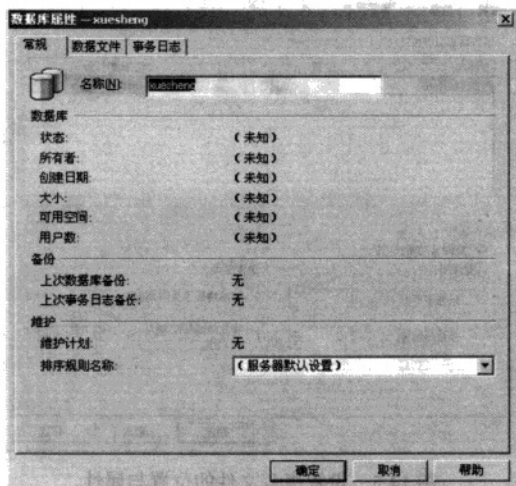


图 14.3 “数据库属性”对话框

(2) 在“常规”选项卡中输入要建立的数据库名称, 在这里要建立一个名称为 xuesheng 的数据库, 故在“名称”文本框中输入 xuesheng。

(3) 切换到“数据文件”选项卡, 设置数据库文件的位置及文件属性。如图 14.4 所示, 表示这个数据库只要有一个主数据文件 (“xuesheng_Data”) 就够了, 而这个数据文件的初始文件大小为 1MB, 可以不断自动增长, 直到占满整个磁盘空间, 每次以 10% 的量增长。

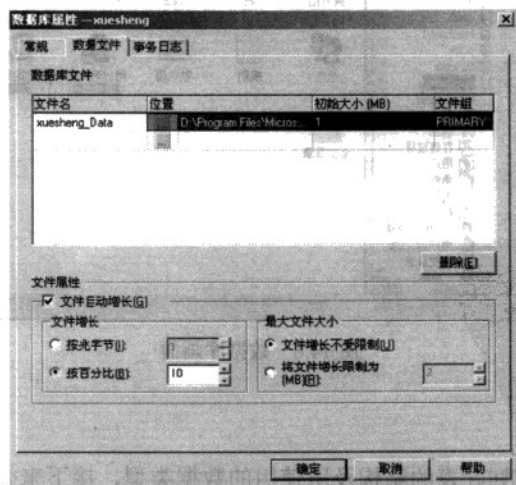


图 14.4 设置数据库文件的位置与属性

(4) 切换到“事务日志”选项卡, 设置事务文件的位置及文件属性。同样也设置这个事务日志文件 xuesheng_Log 的初始文件大小为 1MB, 当数据文件占满了 1MB 就自动扩增 10%

的空间，直到它占满整个磁盘空间，如图 14.5 所示。

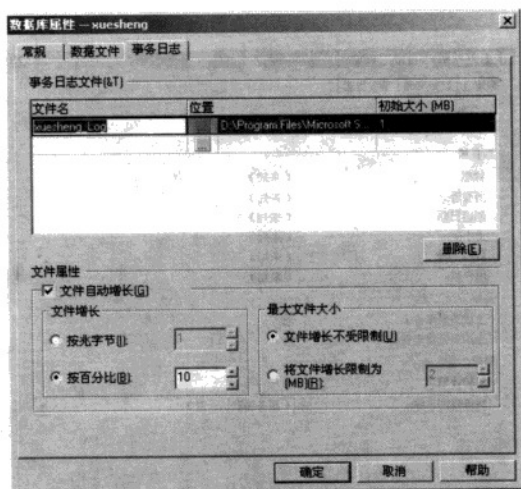


图 14.5 设置事务文件的位置与属性

(5) 最后单击“确定”按钮，开始进行建立数据库及相关数据文件的操作。建立完成后，可以在数据库目录中发现刚才建立的数据库 xuesheng，如图 14.6 所示。

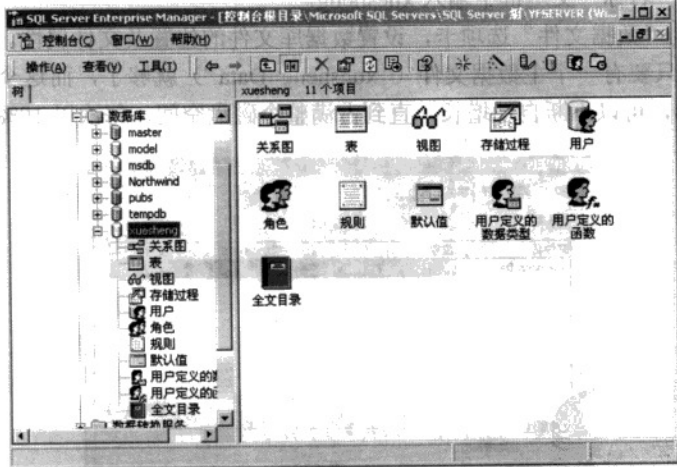


图 14.6 数据库画面

14.3.2 创建表

在 14.2 节中已经规划好表的结构及所使用的数据类型，接下来介绍如何将规划好的表在数据库 xuesheng 中创建起来。

1. 表设计窗口

在 Enterprise Manager 管理工具中，可以利用“表设计”窗口来定义表的列属性与数据特性。“表设计”窗口分为上、下两个窗格，上面的窗格用来定义表中每一个列的主要属性，如

名称、数据类型、长度、是否允许 NULL 等；下方的窗格则可以针对选取的列，定义该列的其他数据属性，如默认值、精度和小数位数等。另外，在窗口上方的工具栏上还有一些工具按钮可以管理表的索引键、定义表间的关系。

下面就来详细介绍该窗口的主要属性的含义。

（1）列的主要属性。“列名”字段：用来惟一标识表内每一个列的名称，指定的名称绝对不能与表内的其他列名相同，也不能是 Transact-SQL 的保留字，而且在指定的名称内不能包含空格或其他特殊符号。

“数据类型”字段：用来指定存放于该表内数据的数据类型，如果要存入的数据与指定的数据类型不符，将会产生类型不符的错误。可以根据要存放信息的数据特性，并根据前面介绍的数据类型说明，为列找到一个适合的数据类型。

“长度”字段：用来指定列的字节数，随着指定的数据类型的不同，这个长度字段所代表的含义也会有所不同。在字符串数据类型中，这个字段值就代表字符串的长度，这时可以修改这个字段值，调整适当的字符串长度来存放数据。对于数值数据类型来说，其所占的字节数是固定的，因此无法更改这个字段的值。

“允许空”字段：用来设置是否允许这个列的内容为空，可以利用鼠标单击的方式选中或取消这个选项。当列被设置为允许为空值时，如果没有为这个列指定任何数据，将会使用空值作为默认值存放到列。反之，如果设置列不允许为空值，就必须自行列为指定默认值，否则在插入数据行时，若未指定任何数据给该列，将会发生错误。

（2）列的其他属性。这些列属性是个别设置到各个列上的，而且随着列定义的数据类型的不同，部分的属性项目可能无法设置。

“描述”字段：这个字段主要提供作为列的批注说明，让管理者日后在维护时，可以更容易了解该列所存放的信息。可以在这个字段中，输入任何字符串文字，当然也可以忽略不填。


“默认值”字段：这个字段是设置当插入数据行而未指定列数据时存入列内的默认值。根据设置的数据类型，这个值可以是文字、数字或是日期时间数据，也可以是一个函数。

“精度”与“小数位数”字段：这两个字段只有在属于 Decimal 或 Numeric 小数数据类型的列中设置，它们分别用来指定整个数值数据的数值位数与小数位数，默认为 (18,0)，表示该列可以存放 18 位数的整数数据（因为小数字数为 0）。在存放分数的列中，分数范围为 0~100，且要保留两位小数位置，可以将“精度”字段设置为 5，“小数位数”字段设置为 2。

“标识”字段：用来设置是否要将列定义为标识列，自动产生列的内容。将列设置为标识字段时，它会自动以“标识种子”字段的内容作为列的初始值，然后以每次增加“标识递增量”的方式，自动为新建的数据行指定该列内容。只有为整数数据类型（如 int、smallint、tinyint）或无小数数字的 decimal 或 numeric 数据类型，可以设置标识选项。

RowGuid 字段：这个字段只有在 uniqueidentifier 数据类型的列中才可以设置，若将它设为“是”，SQL 会自动将 NEWID 函数加入到默认值字段中，使得每当数据加入到表时，列自动取得 GUID 值。

“公式”字段：当列的值与其他列的值有关联时，可以在“公式”字段上输入表达式，自动计算列的内容。

“排序规则”字段：是用来设置列的字符串数据比对、排序方式，可以直接使用默认的“<数据库默认值>”，沿用数据库的排序规则方式；或者是单击字段右方的  按钮，打开列排序


完成上述四个步骤后,就已经初步完成了创建表 stutable 的操作,这时可以直接按下窗口的关闭按钮,关闭设计表。回到控制台窗口后,可以在表的子目录中找到刚创建的表 stutable。


3. 创建表主键

每一个表可以定义一个主键,也可以不定义主键,但不能在一个表上同时定义两个或两个以上的主键。主键主要用于惟一标识表中的每一条记录,使得在定义了主键的数据表中不存在各字段值分别相等的记录,即不存在两条相同的记录。

接下来,就直接以 stutable 表为例来说明如何创建表的主键,步骤如下。

(1) 首先在“表”子目录中找到要设置的表项目 stutable,单击鼠标右键,在打开的快捷菜单中选择“设计表”命令,打开它的“设计表”窗口。

(2) 在上方的表格中选择要定义为主键的列,这里选择“ID”列,然后在工具栏上按下创建主键工具按钮。

(3) 这时可以在列的前面看到钥匙图标,如图 14.9 所示。表示该列已经被定义为表的主键,最后再单击保存工具按钮,让设置真正发生作用。

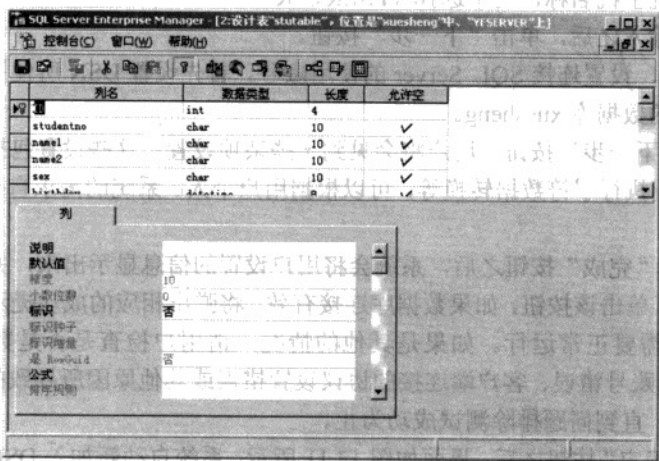


图 14.9 数据行已经被定义为主键

14.3.3 创建实例系统的数据库连接

在 Delphi 2005 中开发数据库应用程序,必须首先和数据库建立连接,它是应用程序和后台数据库交换数据的前提,而这种连接实际上是通过数据源来完成的。这一节介绍如何建立 ODBC 的连接。

现在为前面介绍的数据库 xuesheng 建立 ODBC 数据源,操作步骤如下:

(1) 打开“ODBC 数据源管理器”对话框,选择 User Data Sources 标签页,然后单击“添加”按钮,在出现的 Create New Data Source 对话框中选择 SQL Server 项。

(2) 在用户单击“完成”按钮之后,会见到如图 14.10 所示的对话框。

此对话框包括三个部分:

- Name: 在程序中使用的数据源名称。这里设置数据源名称为 mysqldata。
- Description: 这是为这个系统数据源名称所加注的说明文字。

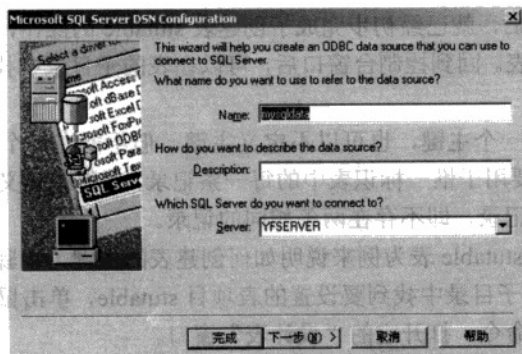


图 14.10 建立新的数据源到 SQL Server

- **Server:** 希望连接的 SQL Server 名称。上图中, 共有两种选项: “(Local)”代表用户当前计算机上的 SQL Server 数据库系统, 其他部分则代表在网络上可以提供连接的数据库系统主机名称, 这里选择 YFSERVER。

在这些选项输入之后, 单击“下一步”按钮。

(3) 接下来, 设置连接 SQL Server 的使用账号。首先设置 DSN 所使用的数据库, 这里选择前面所创建的数据库 xuesheng。

(4) 单击“下一步”按钮, 用户将会见到一些杂项设置。这些设置包括更改 SQL Server 系统信息的语言、执行字符数据转换等, 可以根据用户个人、系统的需求自行设置, 在此采用默认值。

(5) 在单击“完成”按钮之后, 系统会将用户设置的信息显示出来, 并且提供 Test Data Source 测试按钮。单击该按钮, 如果数据库连接有效, 将弹出相应的成功提示信息, 表示 DSN 可以根据用户的需要正常运行。如果是其他的情况, 请用户检查是否是数据库名称错误、Windows NT 登录账号错误、客户端连接的协议设置错误或其他原因所导致的错误, 总之请读者从头逐项检查, 直到问题排除测试成功为止。

(6) 单击“确定”按钮之后, 界面如图 14.11 所示。系统自动新加入 DSN 名称 mysqldata。至此, 数据库 xuesheng 的 ODBC 连接成功创建, 在后面的代码编写中可以直接引用它。

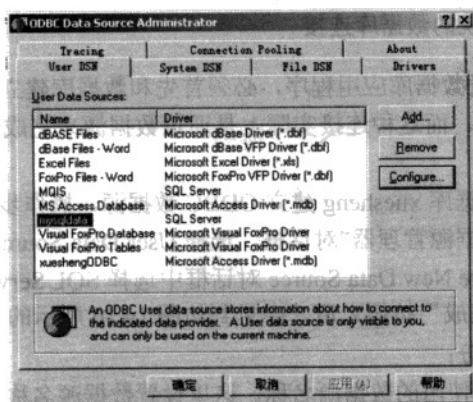


图 14.11 完成后的 ODBC 数据源管理器

14.4 创建实例系统的应用程序

这一节介绍学生信息管理系统应用程序的设计与开发,主要包括主菜单设计、登录模块程序设计、学生信息查询程序设计、学生信息修改程序设计、教师信息管理模块程序设计、课程信息管理模块程序设计、成绩管理模块程序设计和学生个人成绩单打印功能模块设计等。

14.4.1 主程序设计

1. 主菜单设计

主程序设计中的主菜单设计,可以实现对各功能模块的调用,此外还包括工程文件的建立。主程序窗口如图 14.12 所示。

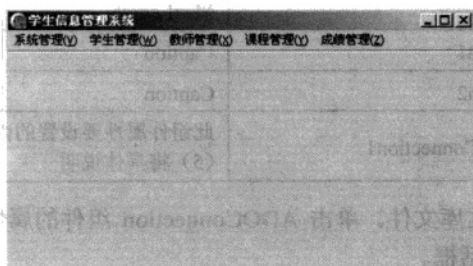


图 14.12 学生信息管理系统主程序窗口

主菜单设计的步骤如下:

(1) 新建工程文件,将工程文件保存为 student.bdsproj,源程序保存为 Main.pas,修改其 Caption 属性值为“学生管理系统”,Name 属性值改为 MainForm。

(2) 向 MainForm 中加入下拉菜单 TMainMenu 控件对象,双击该控件对象,在“系统管理”菜单项下,分别设置菜单条“用户管理”、“修改密码”和“退出”。在“学生管理”菜单项下,分别设置菜单条“学生信息查询”、“学生信息修改”。在“教师管理”菜单项下,设置菜单条“教师信息查询”。在“课程管理”菜单项下,设置菜单条“课程信息查询”。在“成绩管理”菜单项下,分别设置菜单条“成绩信息查询”和“学生个人信息”。

(3) 双击“系统管理”菜单项中的“退出”命令,创建 OnClick 事件,使系统具备退出功能。该功能模块的处理代码如下:

```
procedure TMainForm.N4Click(Sender: TObject);
begin
    Close;
end;
```

2. 登录模块程序设计

系统启动后,将首先出现用户登录窗体,用户先输入用户名,然后输入密码。如果用户三次输入密码错误,将退出程序。

该模块的设置步骤如下:

(1) 新增一个窗体(New Form)。保存源程序名为 checkuser.pas,设置其 Caption 属性值为“用户检测”,设置其 Name 属性值为 CheckUserForm。

(2) 向该窗口中加入两个 TTable 控件对象，两个 TEdit 控件对象和两个 TBitBtn 控件对象，1 个 ADOConnection 控件对象。其 Name 属性值分别为 Lable1、Label2、Edit1、Edit2、BitBtn1 和 BitBtn2 和 ADOConnection1。这些控件对象的其他属性设置如表 14.7 所示。

表 14.7 窗体及各组件属性的设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TTable	Lable1	Caption	用户名
	Lable2	Caption	密码
TEdit	Edit1	Text	空值
		MaxLength	8
	Edit2	Text	空值
		MaxLength	8
TBitBtn	BitBtn1	Caption	确定
	BitBtn2	Caption	取消
ADOConnection	ADOConnection1	此组件属性要设置的内容较多，后续步骤 (3) ~ (5) 将具体说明	

(3) 连接并打开数据库文件。单击 ADOConnection 组件的属性 ConnectionString，可以看见如图 14.13 所示的对话框。

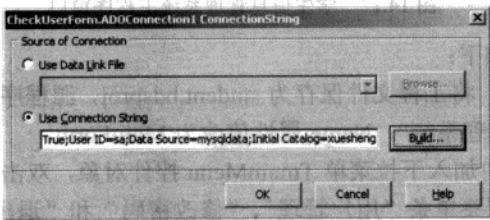


图 14.13 建立连接

(4) 单击 Build 按钮后，打开 Data Link Properties 数据库连接属性编辑对话框，如图 14.14 所示。选中 Use data source name 单选框，并选择其中的数据源 mysqldata；然后在 User name 和 Password 处输入对应的用户名和密码；最后在 Enter the initial catalog to use 处选中即将要用到的数据库 xuesheng。

(5) 单击 Test Connection 按钮，可以测试以上设置是否正确，如果正确，则告知正确信息，若出错则报错，如图 14.15 所示。

(6) 选中该登录窗体，通过双击 Object Inspector 窗口的 Events 页的 OnActivate 栏，创建 OnActivate 事件处理程序，该程序在主要模块启动时执行，通常这里完成初始化功能。该事件的程序处理代码如下：

```
procedure TCheckUserForm.FormActivate(Sender: TObject);
begin
    i:=3;//设置重试次数，i 为全局变量
    edit1.text:='';
```

```

edit2.Text:= "";
edit1.SetFocus;
end;

```

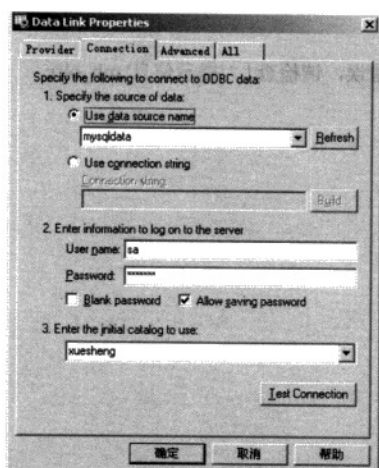


图 14.14 数据库连接属性编辑对话框

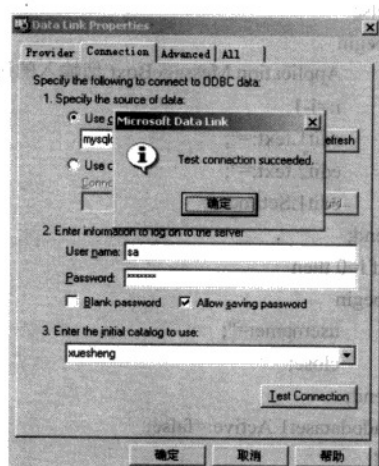


图 14.15 Test Connection 窗口

(7) 双击“确定”按钮，建立该按钮的 OnClick 事件处理程序，即对输入的登录名和密码进行验证。该事件的程序处理代码如下：

```

procedure TCheckUserForm.BitBtn1Click(Sender: TObject);
var Adodataset1:TADODataSet;
begin
  AdoDataSet1:=TADODataSet.Create(self);
  adoDataset1.Connection:=adoconnection1;
  adodataset1.CommandType:=cmdtext;
  Adodataset1.CommandText:=select * from admtable where username1=:username and password1=:password';
  adodataset1.close;
  Adodataset1.Parameters.Clear;
  Adodataset1.Parameters.AddParameter;
  adodataset1.Parameters[0].name:='username';
  adodataset1.Parameters[0].DataType:=ftstring;
  Adodataset1.Parameters[0].Direction:=pdinput;
  adodataset1.Parameters[0].Value:=edit1.text;
  Adodataset1.Parameters.AddParameter;
  adodataset1.Parameters[1].name:='password';
  adodataset1.Parameters[1].DataType:=ftstring;
  Adodataset1.Parameters[1].Direction:=pdinput;
  adodataset1.Parameters[1].Value:=edit2.text;
  adodataset1.active:=true;
  if adodataset1.Recordset.RecordCount=1 then
  begin
    username:=edit1.Text;
    qx[1]:=adodataset1.FieldByName('quanxian1').asboolean;

```

```

        qx[2]:=adodataset1.FieldByName('quanxian2').asboolean;
        close;
    end
    else
    begin
        Application.MessageBox('您输入的用户名或密码错误, 请检查! ', '提示信息', mb_ok);
        i:=i-1;
        edit1.text:="";
        edit2.text:="";
        edit1.SetFocus;
    end;
    if i=0 then
    begin
        username:="";
        close;
    end;
    adodataset1.Active:=false;
end;

```

(8) 双击“取消”按钮, 建立 OnClick 事件, 为该模块设置退出功能。该程序的处理代码如下:

```

procedure TCheckUserForm.BitBtn2Click(Sender: TObject);
begin
    Close;
end;

```

(9) 在主程序 MainForm 中, 为该模块建立事件调用, 通过该模块的 Object Inspector 窗口的 Events 页, 双击 OnActivate 栏, 建立 OnActivate 事件, 该事件在主程序启动时执行。其代码如下:

```

procedure TMainForm.FormActivate(Sender: TObject);
begin
    CheckUserForm.ShowModal();
    if username<>" then
    begin
        if qx[1] then
        begin
            N2.Enabled:=true;
            N3.Enabled:=true;
        end
        else
        begin
            N2.Enabled:=false;
            N3.Enabled:=false;
        end;
        if qx[2] then
        begin
            N6.Enabled:=true;

```

```

N7.Enabled:=true;
N9.Enabled:=true;
N12.Enabled:=true;
N15.Enabled:=true;
N18.Enabled:=true;
end
else
begin
N6.Enabled:=false;
N7.Enabled:=false;
N9.Enabled:=false;
N12.Enabled:=false;
N15.Enabled:=false;
N18.Enabled:=false;
end;
end
else
close;
end;

```

14.4.2 学生信息管理模块程序设计

本节将介绍学生信息管理模块的程序设计，包括学生信息查询程序设计和学生信息修改程序设计。

1. 学生信息查询程序设计

通过此模块来查询学生的信息，提供根据专业、班级等不同方式的查询，该模块的窗口如图 14.16 所示。

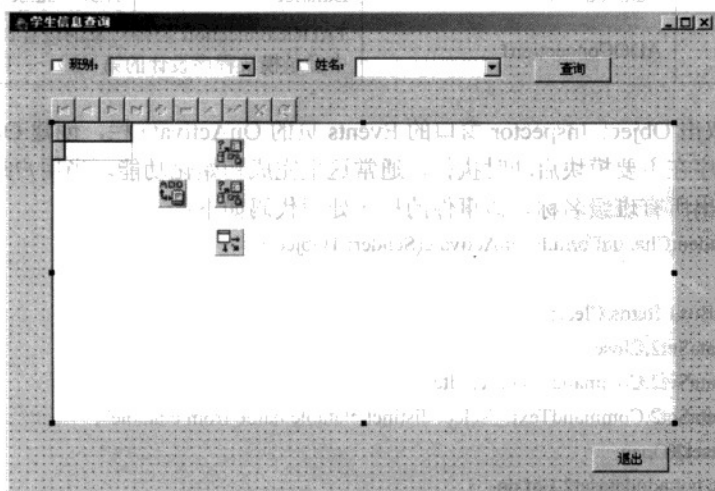


图 14.16 学生信息查询窗口

该模块的设置步骤如下：

(1) 新增一个窗体 (New Form)。保存源程序名为 StudentChaxun.pas, 设置其 Caption 属性值为“学生信息查询”, 设置其 Name 属性值为 StudentChaxunForm。

(2) 向该窗口中加入一个 TDBGrid 控件对象, 一个 TDBNavigator 控件对象、一个 TADOConnection 控件对象、两个 TADODataSet 控件对象、一个 TDataSource 控件对象。这些控件对象的 Name 属性分别设置为 DBGrid1、DBNavigator1、ADOConnection1、ADODataset1、ADODataset2 和 DataSource1。

(3) 向该窗口中加入两个 TCheckBox 控件对象, 两个 TComboBox 控件对象和两个 TButton 控件对象、一个 TDBGrid 控件对象和一个 TDBNavigator 控件对象。这些控件对象的其他属性设置如表 14.8 所示。

表 14.8 学生信息查询中控件对象的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TCheckBox	CheckBox1	Caption	班别:
	CheckBox2	Caption	姓名:
TComboBox	ComboBox1	Caption	(空值)
	ComboBox2	Caption	(空值)
TButton	Button1	Caption	查询
	Button2	Caption	退出
TDBGrid	DBGrid1	DataSource	DataSource1
TDBNavigator	DBNavigator1	DataSource	DataSource1
TADODataSet	ADODataset1	Connection	ADOConnection1
	ADODataset2	Connection	ADOConnection1
TDataSource	DataSource1	DataSet	ADODataset1
ADOConnection	ADOConnection1	TADOConnection 控件对象的设置详见 14.4.1 节中登录模块程序设计的第 (3) ~ (5) 步	

(4) 通过双击 Object Inspector 窗口的 Events 页的 OnActivate 栏, 创建 OnActivate 事件处理程序, 该程序在主要模块启动时执行, 通常这里完成初始化功能。当用户进入该模块时, 触发该事件, 读出所有班级名称。该事件的程序处理代码如下:

```

procedure TStudentChaxunForm.FormActivate(Sender: TObject);
begin
    ComboBox1.Items.Clear;
    ADODataset2.Close;
    ADODataset2.CommandType:=cmdtext;
    ADODataset2.CommandText:='select distinct clatable.name from clatable';
    adodataset2.Open;
    while not adodataset2.Eof do
    begin
        ComboBox1.Items.Add(adodataset2.Fields[0].AsString);
        adodataset2.Next;
    end;
end;

```

end;

end;

(5) 双击“查询”按钮，建立该按钮的 OnClick 事件处理程序。该程序可以让用户选择不同的查询方式以便进行灵活的查询操作。该事件的程序处理代码如下：

```
procedure TStudentChaxunForm.Button1Click(Sender: TObject);
begin
    adodataset1.Close;
    adodataset1.CommandType:=cmdtext;
    if (CheckBox1.Checked=true) and (CheckBox2.Checked=false) then
        adodataset1.commandtext:=
        'select * from stutable,clatable where clatable.name=:classname1 and stutable.classno=clatable.classno ';
        if (CheckBox1.Checked=true) and (CheckBox2.Checked=true) then
            adodataset1.commandtext:=
            'select * from stutable,clatable
            where clatable.name=:classname1 and stutable.classno=clatable.classno and stutable.name1=:name3';
            if (CheckBox1.Checked=false) and (CheckBox2.Checked=false) then
                adodataset1.commandtext:='select * from stutable ';
                ADODataSet1.Parameters.clear;
                adodataset1.Parameters.AddParameter;
                adodataset1.Parameters[0].Name:='classname1';
                adodataset1.Parameters[0].DataType:=ftstring;
                adodataset1.Parameters[0].Direction:=pdinput;
                adodataset1.Parameters[0].Value:= ComboBox1.Text;
                adodataset1.Parameters.AddParameter;
                adodataset1.Parameters[1].Name:='name3';
                adodataset1.Parameters[1].DataType:=ftstring;
                adodataset1.Parameters[1].Direction:=pdinput;
                adodataset1.Parameters[1].Value:= ComboBox2.Text;
                if (CheckBox1.Checked=false) and CheckBox2.Checked then
                    begin
                        adodataset1.commandtext:='select * from stutable where name1=:name4';
                        ADODataSet1.Parameters.clear;
                        adodataset1.Parameters.AddParameter;
                        adodataset1.Parameters[0].Name:='name4';
                        adodataset1.Parameters[0].DataType:=ftstring;
                        adodataset1.Parameters[0].Direction:=pdinput;
                        adodataset1.Parameters[0].Value:= ComboBox2.Text;
                    end;
                adodataset1.Open;
            end;
        end;
    end;
```

end;

(6) 双击“退出”按钮，建立 OnClick 事件，为该模块设置退出功能。该程序的处理代码如下：

```
procedure TStudentChaxunForm.Button2Click(Sender: TObject);
begin
    close;
end;
```

2. 学生信息修改程序设计

该模块主要维护系统中的学生基本信息，它可以对学生基本信息进行增加、删除和修改。该模块设计界面如图 14.17 所示。模块设计步骤如下：

(1) 新建窗体 (New Form)。保存为 StudentXiugai.pas 文件名，设置其 Caption 属性值为“学生信息修改”，设置其 Name 属性值为 StudentXiugaiForm。

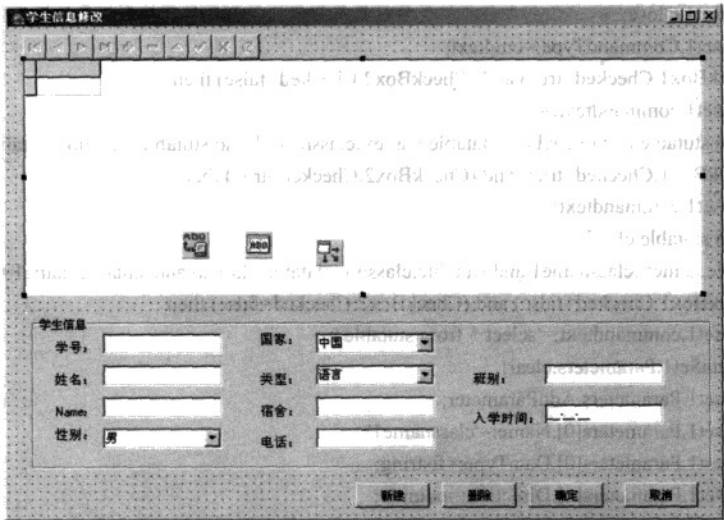


图 14.17 学生信息修改窗口

(2) 向窗口中加入一个 TDBGrid 控件对象、一个 TDBNavigator 控件对象、一个 TADOConnection 控件对象、一个 TADOTable 控件对象、一个 TDataSource 控件对象。这些控件对象的 Name 属性分别设置为 DBGrid1、DBNavigator1、ADOConnection1、ADOTable1 和 DataSource1。

(3) 向窗体中加入一个 TGroupBox 控件对象，其 Name 属性值为 GroupBox1，再向 GroupBox1 控件对象中加入十个 TLabel 控件对象、六个 TEdit 控件对象、三个 TComboBox 控件对象和一个 TMaskEdit 控件对象。

(4) 向窗体中加入四个 TBitBtn 控件对象，以上控件对象的属性设置如表 14.9 所示。

表 14.9 学生信息修改中控件对象的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TGroupBox	GroupBox1	Caption	学生信息
TLabel	Label1	Caption	学号:
	Label2	Caption	姓名:
	Label3	Caption	Name:
	Label4	Caption	性别:
	Label5	Caption	国家:
	Label6	Caption	类型:

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TLabel	Label7	Caption	宿舍:
	Label8	Caption	电话:
	Label9	Caption	班别:
	Label10	Caption	入学时间:
TEdit	Edit1	Text	空值
	Edit2	Text	(空值)
	Edit3	Text	(空值)
	Edit4	Text	(空值)
	Edit5	Text	(空值)
	Edit6	Text	(空值)
TComboBox	ComboBox1	Text	男
	ComboBox2	Text	中国
	ComboBox3	Text	语言
TMaskEdit	MaskEdit1	Text	____
TBitBtn	BitBtn1	Caption	新建
	BitBtn2	Caption	删除
	BitBtn3	Caption	确定
	BitBtn4	Caption	取消
TDBGrid	DBGrid1	DataSource	DataSource1
TDBNavigator	DBNavigator1	DataSource	DataSource1
TADOTable	ADOTable1	Connection	ADOConnection1
TDataSource	DataSource1	DataSet	ADOTable1
ADOConnection	ADOConnection1	TADOConnection 控件对象的设置详见 14.4.1 节中登录模块程序设计的第 (3) ~ (5) 步	

(5) 编写三个函数, 分别为 initiate、SaveToTable 和 LoadFromTable。它们的代码如下:

```

procedure TStudentXiugaiForm.initiate;
begin//用于初始化
    edit1.Text:="";
    edit2.Text:="";
    edit3.Text:="";
    ComboBox1.Text:="";
    ComboBox2.Text:="";
    ComboBox3.Text:="";
    edit7.Text:="";
    edit8.Text:="";
    edit10.Text:="";
    maskedit1.Text:=DateToStr(date());

```



```

edit1.SetFocus;
end;

procedure TStudentXiugaiForm.SaveToTable;
begin//保存当前信息
    adotable1.edit;
    adotable1.FieldByName('studentno').asString:=edit1.text;
    adotable1.FieldByName('name1').AsString:=edit2.text;
    adotable1.FieldByName('name2').AsString:=edit3.text;
    adotable1.FieldByName('sex').asString:=ComboBox1.text;
    adotable1.FieldByName('country').asString:=ComboBox2.text;
    adotable1.FieldByName('type').asString:=ComboBox3.text;
    adotable1.FieldByName('dorm').asString:=edit7.text;
    adotable1.FieldByName('tel1').asString:=edit8.text;
    adotable1.FieldByName('classno').asString:=edit10.text;
    adotable1.FieldByName('intime').asdatetime:=StrToDate(maskedit1.text);
    adotable1.Post;
end;

```

```

procedure TStudentXiugaiForm.LoadFromTable;
begin//从当前已打开的数据表中读取指针所指的记录
    edit1.text:=adotable1.FieldByName('studentno').asString;
    edit2.text:=adotable1.FieldByName('name1').asString;
    edit3.text:=adotable1.FieldByName('name2').AsString;
    ComboBox1.text:=adotable1.FieldByName('sex').asString;
    ComboBox2.Text:=adotable1.FieldByName('country').asString;
    ComboBox3.text:=adotable1.FieldByName('type').asString;
    edit7.Text:=adotable1.FieldByName('dorm').asString;
    edit8.Text:=adotable1.FieldByName('tel1').asString;
    edit10.Text:=adotable1.FieldByName('classno').asString;
    maskedit1.text:=DateToStr(adotable1.FieldByName('intime').asdatetime);
end;

```

(6) 为 Edit1 控件对象创建 OnExit 事件, 该事件在光标离开 Edit1 对象时触发, 主要检测在新增学生信息时, 输入的学号是否和已有的重复, 以及在修改和删除操作时输入的班级名是否存在等。该事件的处理程序代码如下:

```

procedure TStudentXiugaiForm.Edit1Exit(Sender: TObject);
var
    adodataset1:TAdoDataset;

begin
    if edit1.text<>"" then
        begin
            adodataset1:=TAdoDataset.Create(self);
            adodataset1.Connection:=adoconnection1;
            adodataset1.Close;
            adodataset1.CommandType:=cmdText;

```

```

adodataset1.CommandText:='select * from stutable where studentno=:studentno1';
adodataset1.Parameters.clear;
adodataset1.Parameters.AddParameter;
adodataset1.Parameters[0].Name:='studentno1';
adodataset1.Parameters[0].DataType:=ftstring;
adodataset1.Parameters[0].Direction:=pdinput;
adodataset1.Parameters[0].Value:=edit1.text;
adodataset1.active:=true;
if new_record then
begin
    if adodataset1.Recordset.RecordCount=1 then
    begin
        application.MessageBox('您输入学号错误, 请检查!', '提示信息', mb_ok);
        edit1.SetFocus;
    end
end;
adodataset1.close;
end;
end;

```

(7) 创建 OnActivate 事件处理程序, 该程序在模块启动时执行, 完成初始化功能。在 Object Inspector 中选择该 Form 的 Event 页, 单击其中的 OnActivate 便可编写代码, 该事件处理程序代码如下:

```

procedure TStudentXiugaiForm.FormActivate(Sender: TObject);
begin
    dbgrid1.DataSource:=datasource1;
    datasource1.DataSet:=adotable1;
    adotable1.Connection:=adoconnection1;
    adotable1.tableName:='stutable';
    adotable1.active:=true;
    initiate;
    new_record:=false;
    if adotable1.Eof then
        bitbtn2.Enabled:=false
    else
        loadfromtable;
end;

```

(8) 为“确定”按钮建立 OnClick 事件, 该事件处理程序代码如下:

```

procedure TStudentXiugaiForm.BitBtn3Click(Sender: TObject);
begin
    if new_record then
    begin
        adotable1.Append;
        new_record:=false;
    end;
    savetotable;
    if adotable1.Recordset.RecordCount>0 then
        bitbtn2.Enabled:=true;
end;

```

(9) 为“删除”按钮建立 OnClick 事件，该事件处理程序代码如下：

```
procedure TStudentXiugaiForm.BitBtn2Click(Sender: TObject);
```

```
begin
```

```
    adotable1.Delete;
```

```
    adotable1.first;
```

```
end;
```

(10) 为“新增”按钮建立 OnClick 事件，该事件处理程序代码如下：

```
procedure TStudentXiugaiForm.BitBtn1Click(Sender: TObject);
```

```
begin
```

```
    initiate;
```

```
    new_record:=true;
```

```
end;
```

(11) 为“取消”按钮建立 OnClick 事件，该事件处理程序代码如下：

```
procedure TStudentXiugaiForm.BitBtn4Click(Sender: TObject);
```

```
begin
```

```
    close;
```

```
end;
```

(12) 在 MainForm 中建立调用本模块的事件。在 MainForm 中双击 MainMenu1 对象，在出现的菜单中选中“学生信息修改”窗体，并双击它，便可编写代码。该事件处理程序代码如下：

```
procedure TMainForm.N7Click(Sender: TObject);
```

```
begin
```

```
    StudentXiugaiForm.ShowModal();
```

```
end;
```

14.4.3 教师信息管理模块程序设计

该模块主要实现教师信息的查询，打开该模块之后，在第一个数据集中显示所有教师的基本信息；当在第一个数据集中选择不同的教师时，在第二个数据集中则显示该选中教师的任课情况。该模块设计界面如图 14.18 所示。

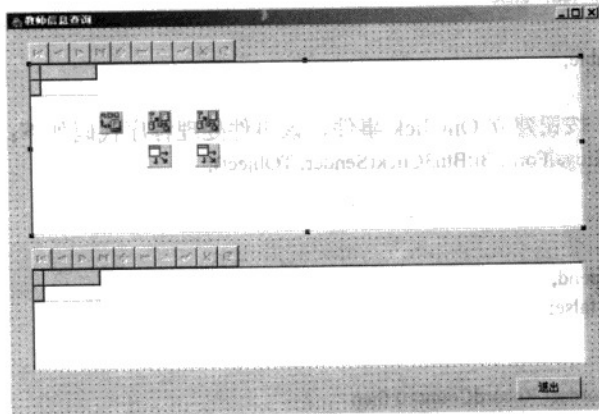


图 14.18 教师信息管理窗口

该模块设计步骤如下:

(1) 新建窗体 (New Form)。保存为 Teacher.pas 文件名, 设置其 Caption 属性值为“教师信息查询”, 设置其 Name 属性值为 TeacherForm。

(2) 向窗口中加入两个 TDBGrid 控件对象、两个 TDBNavigator 控件对象、一个 TADOConnection 控件对象、两个 TADODataset 控件对象、两个 TDataSource 控件对象。这些控件对象的 Name 属性分别设置为 DBGrid1、DBGrid2、DBNavigator1、DBNavigator2、ADOConnection1、ADODataset1、ADODataset2、DataSource1 和 DataSource2。

(3) 向窗体中加入一个 TButton 控件对象, 以上控件对象的属性设置如表 14.10 所示。

表 14.10 教师信息管理中控件的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TButton	Button1	Caption	退出
TDBGrid	DBGrid1	DataSource	DataSource1
	DBGrid2	DataSource	DataSource2
TDBNavigator	DBNavigator1	DataSource	DataSource1
	DBNavigator2	DataSource	DataSource2
TADODataset	ADODataset1	Connection	ADOConnection1
	ADODataset2	Connection	ADOConnection1
TDataSource	DataSource1	DataSet	ADODataset1
	DataSource2	DataSet	ADODataset2
ADOConnection	ADOConnection1	TADOConnection 控件对象的设置详见 14.4.1 节中登录模块程序设计的第 (3) ~ (5) 步	

(4) 选中该教师信息查询窗体, 通过双击 Object Inspector 窗口的 Events 页的 OnActivate 栏, 创建 OnActivate 事件处理程序, 该程序在主要模块启动时执行, 完成初始化。该事件的程序处理代码如下:

```
procedure TTeacherForm.FormActivate(Sender: TObject);
begin
    adodataset1.Close;
    adodataset1.CommandType:=cmdtext;
    adodataset1.commandtext:='select * from teatable';
    adodataset1.Open;
end;
```

(5) 选中控件对象 ADODataset1, 通过双击 Object Inspector 窗口的 Events 页的 AfterScroll 栏, 创建 ADODataset1AfterScroll 事件处理程序, 当记录集的当前记录指针改变时, 触发该事件。该事件的程序处理代码如下:

```
procedure TTeacherForm.ADODataset1AfterScroll(DataSet: TDataSet);
begin
    adodataset2.Close;
    adodataset2.CommandType:=cmdtext;
    adodataset2.commandtext:='select * from coutable';
end;
```

```

where teacherno = "" + adodataset1.Fields[1].AsString + "";
adodataset2.Open;
end;

```

(6) 为“退出”按钮建立 OnClick 事件, 该事件处理程序代码如下:

```

procedure TTeacherForm.Button3Click(Sender: TObject);
begin
close;
end;

```

(7) 在 MainForm 中建立调用本模块的事件。在 MainForm 中双击 MainMenu1 对象, 在出现的菜单中选中“教师信息查询”窗体, 双击它便可编写代码。该事件处理程序代码如下:

```

procedure TMainForm.N9Click(Sender: TObject);
begin
TeacherForm.ShowModal();
end;

```

14.4.4 课程信息管理模块程序设计

该模块主要实现课程信息的查询。打开该模块之后, 在两个 TComboBox 控件对象的下列表中分别显示所有班级和所有课程给用户选择; 并且用户可以根据两个 TCheckBox 控件对象的选择情况进行灵活的查询; 第一个数据集中显示被选条件下的查询结果, 当在第一个数据集中选择不同的课程时, 在第二个数据集中则显示该选中课程的任课教师情况, 以便用户掌握课程和教师之间的关系。该模块设计界面如图 14.19 所示。

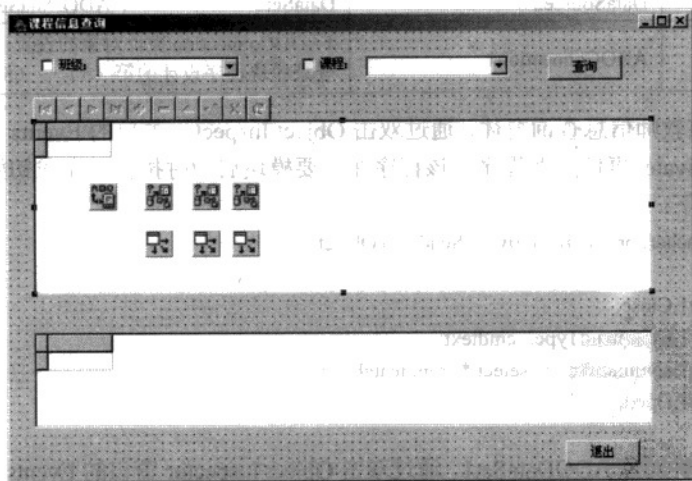


图 14.19 课程信息管理窗口

该模块设计步骤如下:

(1) 新建窗体 (New Form)。保存为 KechengChaxun.pas 文件名, 设置其 Caption 属性值为“课程信息查询”, 设置其 Name 属性值为 KechengChaxunForm。

(2) 向窗口中加入两个 TDBGrid 控件对象、一个 TDBNavigator 控件对象、一个 TADOConnection 控件对象、三个 TADODataset 控件对象、三个 TDataSource 控件对象。这些

控件对象的 Name 属性分别设置为 DBGrid1、DBGrid2、DBNavigator1、ADOConnection1、ADODataset1、ADODataset2、ADODataset3、DataSource1、DataSource2 和 DataSource3。

(3) 向窗体中加入两个 TButton 控件对象、两个 TCheckBox 控件对象和两个 TComboBox 控件对象。以上控件对象的属性设置如表 14.11 所示。

表 14.11 课程信息管理中控件对象的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TButton	Button1	Caption	查询
	Button2	Caption	退出
TCheckBox	CheckBox1	Caption	班级:
	CheckBox2	Caption	课程:
TComboBox	ComboBox1	Text	(空值)
	ComboBox2	Text	(空值)
TDBGrid	DBGrid1	DataSource	DataSource2
	DBGrid2	DataSource	DataSource3
TDBNavigator	DBNavigator1	DataSource	DataSource2
TADODataset	ADODataset1	Connection	ADOConnection1
	ADODataset2	Connection	ADOConnection1
	ADODataset3	Connection	ADOConnection1
TDataSource	DataSource1	DataSet	ADODataset1
	DataSource2	DataSet	ADODataset2
	DataSource3	DataSet	ADODataset3
ADOConnection	ADOConnection1	TADOConnection 控件对象的设置详见 14.4.1 节中登录模块程序设计的第 (3) ~ (5) 步	

(4) 选中该课程信息查询窗体, 通过双击 Object Inspector 窗口的 Events 页的 OnActivate 栏, 创建 OnActivate 事件处理程序, 该程序在主要模块启动时执行, 完成初始化。该事件处理程序代码如下:

```

procedure TKechengChaxunForm.FormActivate(Sender: TObject);
begin
    ComboBox1.Items.Clear;
    ADODataset1.Close;
    ADODataset1.CommandType:=cmdtext;
    ADODataset1.CommandText:='select distinct clatable.name from clatable';
    adodataset1.Open;
    while not adodataset1.Eof do
    begin
        ComboBox1.Items.Add(adodataset1.Fields[0].AsString);
        adodataset1.Next;
    end;
    ComboBox2.Items.Clear;

```

```

ADODataset1.Close;
ADODataset1.CommandType:=cmdtext;
ADODataset1.CommandText:='select distinct coutable.name from coutable' ;
adodataset1.Open;
while not adodataset1.Eof do
begin
    ComboBox2.Items.Add(adodataset1.Fields[0].AsString);
    adodataset1.Next;
end;
end;

```

(5) 双击“查询”按钮，创建查询处理程序。该事件处理程序代码如下：

```

procedure TKechengChaxunForm.Button1Click(Sender: TObject);
begin
    adodataset2.Close;
    adodataset2.CommandType:=cmdtext;
    if (CheckBox1.Checked=true) and (CheckBox2.Checked=false) then
        adodataset2.commandtext:='select * from coutable where classname=:classname1';
    if (CheckBox1.Checked=true) and (CheckBox2.Checked=true) then
        adodataset2.commandtext:='select * from coutable
            where classname=:classname1 and name=:name1';
    if (CheckBox1.Checked=false) and (CheckBox2.Checked=false) then
        adodataset2.commandtext:='select * from coutable';
    ADODataset2.Parameters.clear;
    adodataset2.Parameters.AddParameter;
    adodataset2.Parameters[0].Name:='classname1';
    adodataset2.Parameters[0].DataType:=ftstring;
    adodataset2.Parameters[0].Direction:=pdinput;
    adodataset2.Parameters[0].Value:= ComboBox1.Text;
    adodataset2.Parameters.AddParameter;
    adodataset2.Parameters[1].Name:='name1';
    adodataset2.Parameters[1].DataType:=ftstring;
    adodataset2.Parameters[1].Direction:=pdinput;
    adodataset2.Parameters[1].Value:= ComboBox2.Text;
    if (CheckBox1.Checked=false) and CheckBox2.Checked then
        begin
            adodataset2.commandtext:='select * from coutable where name=:name2';
            ADODataset2.Parameters.clear;
            adodataset2.Parameters.AddParameter;
            adodataset2.Parameters[0].Name:='name2';
            adodataset2.Parameters[0].DataType:=ftstring;
            adodataset2.Parameters[0].Direction:=pdinput;
            adodataset2.Parameters[0].Value:= ComboBox2.Text;
        end;
    adodataset2.Open;
end;

```

(6) 选中控件对象 ADODataset2，通过双击 Object Inspector 窗口的 Events 页的 AfterScroll

栏, 创建 ADODataset1AfterScroll 事件处理程序, 当记录集的当前记录指针改变时, 触发该事件。该事件处理程序代码如下:

```
procedure TKchengChaxunForm.ADODataset2AfterScroll(DataSet: TDataSet);  
begin  
    adodataset3.Close;  
    adodataset3.CommandType:=cmdtext;  
    adodataset3.commandtext:='select * from teatable where teatable.teacherno="'+  
    adodataset2.Fields[9].AsString+'";  
    adodataset3.Open;  
end;
```

(7) 为“退出”按钮建立 OnClick 事件, 该事件处理程序代码如下:

```
procedure TTeacherForm.Button3Click(Sender: TObject);  
begin  
    close;  
end;
```

(8) 在 MainForm 中建立调用本模块的事件。在 MainForm 中双击 MainMenu1 对象, 在出现的菜单中选中“课程信息查询”窗体, 并双击它, 便可编写代码。该事件处理程序代码如下:

```
procedure TMainForm.N12Click(Sender: TObject);  
begin  
    KchengChaxunForm.ShowModal();  
end;
```

14.4.5 成绩管理模块程序设计

该模块实现成绩信息查询。用户打开该模块, 控件对象 ComboBox1 的下拉列表中则显示所有班级提供给用户选择; 当用户在 ComboBox1 中选中某个班级的时候, 在控件对象 ComboBox2 的下拉列表中则显示相应的课程名称, 以供用户选择, 成绩管理窗口如图 14.20 所示。

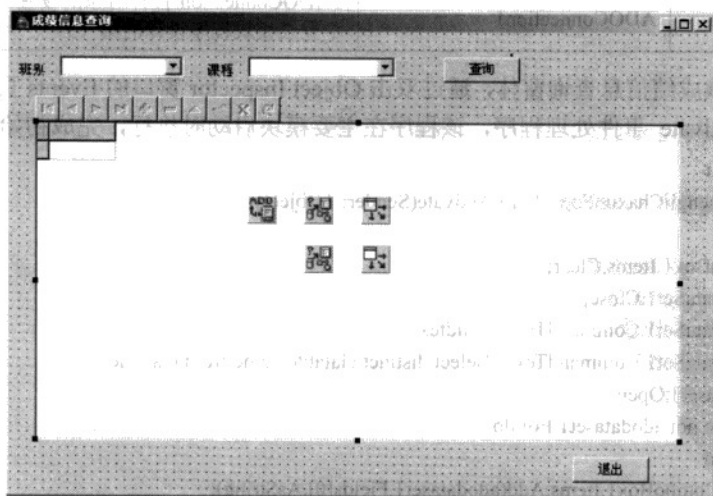


图 14.20 成绩管理窗口

模块设计步骤如下:

(1) 新建窗体 (New Form)。保存为 ChengjiChaxun.pas 文件名, 设置其 Caption 属性值为“成绩信息查询”, 设置其 Name 属性值为 ChengjiChaxunForm。

(2) 向窗口中加入一个 TDBGrid 控件对象、一个 TDBNavigator 控件对象、一个 TADOConnection 控件对象、两个 TADODataSet 控件对象、两个 TDataSource 控件对象。这些控件对象的 Name 属性分别设置为 DBGrid1、DBNavigator1、ADOConnection1、ADODataset1、ADODataset2、DataSource1 和 DataSource2。

(3) 向窗体中加入两个 TButton 控件对象、两个 TLabel 控件对象和两个 TComboBox 控件对象。以上控件对象的属性设置如表 14.12 所示。

表 14.12 成绩管理中控件对象的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TButton	Button1	Caption	查询
	Button2	Caption	退出
TLabel	Label1	Caption	班别:
	Label2	Caption	课程:
TComboBox	ComboBox1	Text	空值
	ComboBox2	Text	空值
TDBGrid	DBGrid1	DataSource	DataSource2
TDBNavigator	DBNavigator1	DataSource	DataSource2
TADODataSet	ADODataset1	Connection	ADOConnection1
	ADODataset2	Connection	ADOConnection1
TDataSource	DataSource1	DataSet	ADODataset1
	DataSource2	DataSet	ADODataset2
ADOConnection	ADOConnection1	TADOConnection 控件对象的设置详见 14.4.1 节中登录模块程序设计的第 (3) ~ (5) 步	

(4) 选中该成绩信息查询窗体, 通过双击 Object Inspector 窗口的 Events 页的 OnActivate 栏, 创建 OnActivate 事件处理程序, 该程序在主要模块启动时执行, 完成初始化。该事件处理程序代码如下:

```

procedure TChengjiChaxunForm.FormActivate(Sender: TObject);
begin
    ComboBox1.Items.Clear;
    ADODataset1.Close;
    ADODataset1.CommandType:=cmdtext;
    ADODataset1.CommandText:='select distinct clatable.name from clatable';
    adodataset1.Open;
    while not adodataset1.Eof do
    begin
        ComboBox1.Items.Add(adodataset1.Fields[0].AsString);
        adodataset1.Next;
    end;
end;

```

end;

end;

(5) 双击“查询”按钮，创建查询处理程序。该事件的程序处理代码如下：

```
procedure TChengjiChaxunForm.Button1Click(Sender: TObject);
begin
    adodataset2.Close;
    adodataset2.CommandType:=cmdtext;
    adodataset2.commandtext:='select * from restable
        where classname=:classname1 and coursenam=:coursenamel';
    ADODataSet2.Parameters.clear;
    adodataset2.Parameters.AddParameter;
    adodataset2.Parameters[0].Name:='classname1';
    adodataset2.Parameters[0].DataType:=ftstring;
    adodataset2.Parameters[0].Direction:=pdinput;
    adodataset2.Parameters[0].Value:= ComboBox1.Text;
    adodataset2.Parameters.AddParameter;
    adodataset2.Parameters[1].Name:='coursenamel';
    adodataset2.Parameters[1].DataType:=ftstring;
    adodataset2.Parameters[1].Direction:=pdinput;
    adodataset2.Parameters[1].Value:= ComboBox2.Text;
    adodataset2.Open;
end;
```

(6) 选中控件对象 ComboBox1，通过双击 Object Inspector 窗口的 Events 页的 OnChange 栏，创建 ComboBox1Change 事件处理程序，当 ComboBox1 中选中的班别改变时，触发该事件，使得 ComboBox2 中出现与该班别相关的课程。该事件处理程序代码如下：

```
procedure TChengjiChaxunForm.ComboBox1Change(Sender: TObject);
begin
    ComboBox2.Items.Clear;
    ADODataSet1.Close;
    ADODataSet1.CommandType:=cmdtext;
    ADODataSet1.CommandText:='select distinct coutable.name
        from coutable,restable
        where restable.classname=:s0 and
            restable.courseno=coutable.courseno';

    ADODataSet1.Parameters.clear;
    adodataset1.Parameters.AddParameter;
    adodataset1.Parameters[0].Name:='s0';
    adodataset1.Parameters[0].DataType:=ftstring;
    adodataset1.Parameters[0].Direction:=pdinput;
    adodataset1.Parameters[0].Value:= ComboBox1.Text;
    adodataset1.Open;
    while not adodataset1.Eof do
    begin
        ComboBox2.Items.Add(adodataset1.Fields[0].AsString);
        adodataset1.Next;
    end;
end;
```

end;

(7) 为“退出”按钮建立 OnClick 事件, 该事件处理程序代码如下:

```
procedure TChengjiChaxunForm.Button2Click(Sender: TObject);
begin
close;
end;
```

(8) 在 MainForm 中建立调用本模块的事件。在 MainForm 中双击 MainMenu1 对象, 在出现的菜单中选中“成绩信息查询”窗体, 并双击它, 便可编写代码。该事件处理程序代码如下:

```
procedure TMainForm.N15Click(Sender: TObject);
begin
ChengjiChaxunForm.ShowModal();
end;
```

14.4.6 学生个人成绩单打印功能模块设计

该模块主要实现学生个人成绩查询与成绩单打印功能, 该模块的窗口如图 14.21 所示。

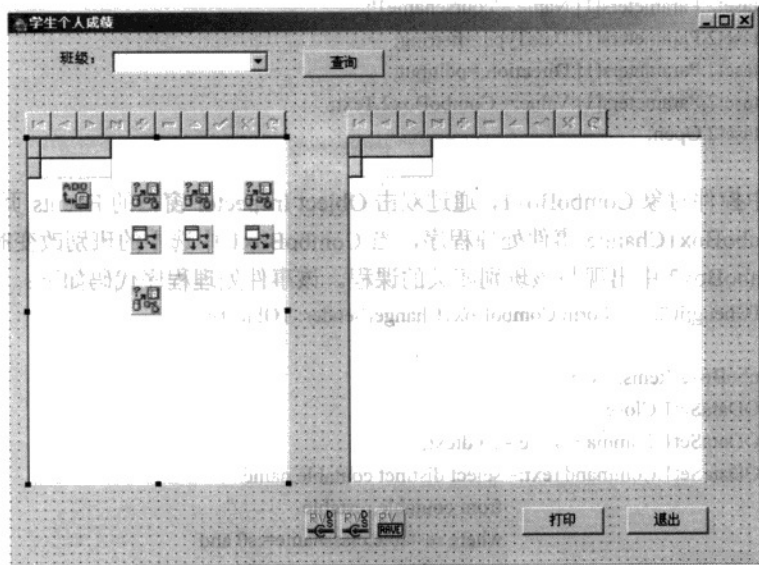


图 14.21 学生个人成绩单打印窗口

该模块设计步骤如下:

(1) 新建窗体 (New Form)。保存为 ChengjiGeren.pas 文件名, 设置其 Caption 属性值为“学生个人成绩”, 设置其 Name 属性值为 ChengjiGerenForm。

(2) 向窗口中加入两个 TDBGrid 控件对象、两个 TDBNavigator 控件对象、一个 TADOConnection 控件对象、四个 TADODataSet 控件对象、三个 TDataSource 控件对象。这些控件对象的 Name 属性分别设置为 DBGrid1、DBGrid2、DBNavigator1、DBNavigator2、ADOConnection1、ADODataset1、ADODataset2、ADODataset3、ADODataset4、DataSource1、DataSource2 和 DataSource3。

(3) 向窗体中加入三个 TButton 控件对象、一个 TLabel 控件对象和一个 TComboBox 控件对象。以上控件对象的属性设置如表 14.13 所示。

表 14.13 学生个人成绩单打印模块中控件对象的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
TButton	Button1	Caption	查询
	Button2	Caption	退出
	Button3	Caption	打印
TLabel	Label1	Caption	班级:
TComboBox	ComboBox1	Text	(空值)
TDBGrid	DBGrid1	DataSource	DataSource1
	DBGrid2	DataSource	DataSource2
TDBNavigator	DBNavigator1	DataSource	DataSource1
	DBNavigator2	DataSource	DataSource2
TADODataSet	ADODataset1	Connection	ADODataset1
	ADODataset2	Connection	ADODataset1
	ADODataset3	Connection	ADODataset1
	ADODataset4	Connection	ADODataset1
TDataSource	DataSource1	DataSet	ADODataset1
	DataSource2	DataSet	ADODataset2
	DataSource3	DataSet	ADODataset3
TRvDataSetConnection	RvDataSetConnection1	DataSet	ADODataset3
	RvDataSetConnection2	DataSet	ADODataset4
TRvProject	RvProject1	ProjectFile	Project1.rav
ADODataset	ADODataset1	TADODataset 控件对象的设置详见 14.4.1 节中登录模块程序设计的第 (3) ~ (5) 步	

(4) 选中该个人成绩查询窗体, 通过双击 Object Inspector 窗口的 Events 页的 OnActivate 栏, 创建 OnActivate 事件处理程序, 该程序在主要模块启动时执行, 完成初始化。该事件处理程序代码如下:

```

procedure TChengjiGerenForm.FormActivate(Sender: TObject);
begin
    ComboBox1.Items.Clear;
    ADODataset1.Close;
    ADODataset1.CommandType:=cmdtext;
    ADODataset1.CommandText:='select distinct clatable.name from clatable';
    adodataset1.Open;
    while not adodataset1.Eof do
    begin
        ComboBox1.Items.Add(adodataset1.Fields[0].AsString);
        adodataset1.Next;
    end;
end;

```

end;

(5) 双击“查询”按钮, 创建查询处理程序。该事件的程序处理代码如下:

```
procedure TChengjiGerenForm.Button1Click(Sender: TObject);
begin
    adodataset2.Close;
    adodataset2.CommandType:=cmdtext;
    adodataset2.commandtext:='select distinct studentno,studentname
                                from restable where classname=:classname1';
    ADODataSet2.Parameters.clear;
    adodataset2.Parameters.AddParameter;
    adodataset2.Parameters[0].Name:='classname1';
    adodataset2.Parameters[0].DataType:=ftstring;
    adodataset2.Parameters[0].Direction:=pdinput;
    adodataset2.Parameters[0].Value:= ComboBox1.Text;
    adodataset2.Open;
end;
```

(6) 选中控件对象 ADODataSet2, 通过双击 Object Inspector 窗口的 Events 页的 AfterScroll 栏, 创建 ADODataSet2AfterScroll 事件处理程序, 当在左边数据集选中某个学生时, 触发该事件, 使得右边数据集中出现与该学生相关的个人成绩信息。该事件的程序处理代码如下:

```
procedure TChengjiGerenForm.ADODataSet2AfterScroll(DataSet: TDataSet);
begin
    adodataset3.Close;
    adodataset3.CommandType:=cmdtext;
    adodataset3.commandtext:='select * from restable
                                where studentno="'+adodataset2.Fields[0].AsString+'"order by time asc';
    adodataset3.Open;
    adodataset4.Close;// adodataset4 被用于打印功能
    adodataset4.CommandType:=cmdtext;
    adodataset4.commandtext:='select * from stutable
                                where studentno="'+adodataset2.Fields[0].AsString+'";
    adodataset4.Open;
end;
```

(7) 选中控件对象 RvDataSetConnection1, 通过双击 Object Inspector 窗口的 Properties 页的 DataSet 项中的 CommandText, 将出现一个 CommandTextEditor 窗口, 在其中输入如图 14.22 所示的 SQL 语句。

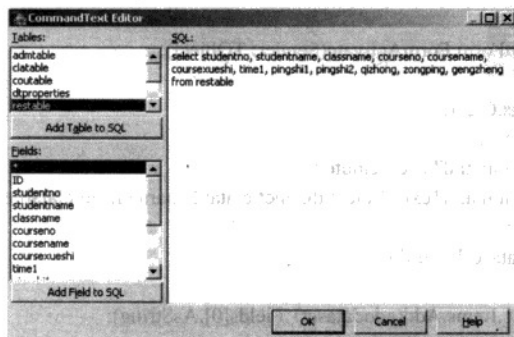


图 14.22 RvDataSetConnection1 的设置

以类似的方法对控件对象 RvDataSetConnection2 进行设置, 如图 14.23 所示。

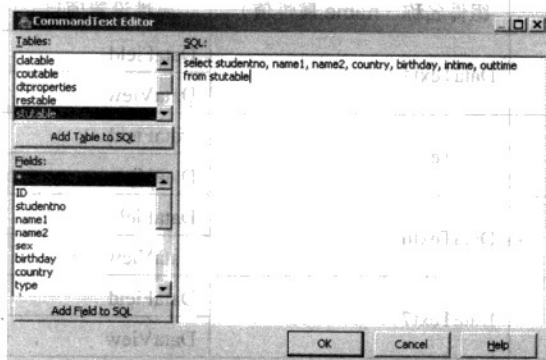


图 14.23 RvDataSetConnection2 的设置

(8) 双击控件对象 RvProject1, 进入 Rave Reports 6 报表设计器, 在其中加入如表 14.14 所示的组件。

表 14.14 控件对象 RvProject1 中组件的属性设置

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
Band component	DataView1TitleBand	ControllerBand	DataView1DataBand
Band component	DataView1Band	ControllerBand	DataView1DataBand
DataBand component	DataView1DataBand	DataView	DataView1
DataBand component	Band1	Visible	True
Text component	TitleText	Text	学生考试成绩单
Text component	Text1	Text	国籍:
Text component	Text2	Text	出生日期:
Text component	Text3	Text	课程
Text component	Text4	Text	分数
Text component	Text5	Text	姓名:
Text component	Text6	Text	学习时间:
Text component	Text7	Text	至
Text component	Text8	Text	国际教育学院
Text component	Text9	Text	总学时
Text component	Text10	Text	学期
DataText component	DataText1	DataField	name2
		DataView	DataView2
DataText component	DataText2	DataField	country
		DataView	DataView2
DataText component	DataText3	DataField	Coursename
		DataView	DataView1

续表

组件类别	组件名称 (name 属性值)	属性设置项目	设置结果
DataText component	DataText4	DataField	pingshi2
		DataView	DataView1
DataText component	DataText5	DataField	name1
		DataView	DataView2
DataText component	DataText6	DataField	Birthday
		DataView	DataView2
DataText component	DataText7	DataField	Intime
		DataView	DataView2
DataText component	DataText8	DataField	Outtime
		DataView	DataView2
DataText component	DataText9	Mirror	DataText8
DataText component	DataText10	DataField	Coursexueshi
		DataView	DataView1
DataText component	DataText11	DataField	time1
		DataView	DataView1

然后，按照此表设置各个组件的属性，将设计出如图 14.24 所示的报表。

▼ DataView1Region: DataView1TitleBand (BGRD788 1PC)

学生考试成绩单

姓名: [name1] | [name2] |

国籍: [country] | 出生日期: [birthday] |

学习时间: [intime] | 至 [outtime] |

▼ DataView1Region: DataView1Band (BGRD788 1PC)

课程 | 总学时 | 分数 | 学期

◆ DataView1Region: DataView1DataBand (Master 1PC)

[course name] | [course] | [qimo] | [time]

◆ DataView1Region: Band1 (Master 1PC)

国际教育学院

[outtime]

▼ DataView1Region: DataView1PageBand (BGRD788 1PC)

图 14.24 学生考试成绩单报表设计窗口

其中，图 14.24 和表 14.14 中的各个组件的组织关系如图 14.25 所示。

(9) 为“打印”按钮建立OnClick事件，调用已经完成设计的报表 RvProject1。该事件处理程序代码如下：

```
procedure TChengjiGerenForm.Button4Click(Sender: TObject);
begin
```

```
RvProject1.Execute;
```

```
end;
```

(10) 为“退出”按钮建立 OnClick 事件, 该事件处理程序代码如下:

```
procedure TChengjiGerenForm.Button2Click(Sender: TObject);
begin
    close;
end;
```

(11) 在 MainForm 中建立调用本模块的事件。在 MainForm 中双击 MainMenu1 对象, 在出现的菜单中选“成绩信息查询”窗体, 并双击它, 便可编写代码。该事件处理程序代码如下:

```
procedure TMainForm.N15Click(Sender: TObject);
begin
    ChengjiChaxunForm.ShowModal();
end;
```

运行实例程序, 单击“打印”按钮时, 则会弹出一个输出选项窗口。然后, 从中选择 Preview 选项, 则可以预览报表, 如图 14.26 所示; 如果选择 Printer 选项, 则可以将报表输出到打印机中; 如果选择 File 选项并选择保存的文件名称, 则会将报表保存到该文件中。

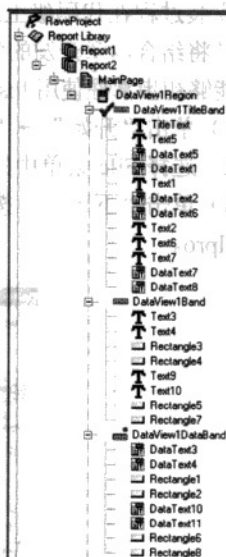


图 14.25 各个组件的组织关系

课程	总学时	分数	学期
中国文学	40	90	2005
西方文学	50	88	2005
中国武术	30	98	2005

图 14.26 预览报表窗口

14.5 实例系统的安装程序制作

InstallShield 是一种非常成功的应用软件安装程序制作工具, 以其功能强大、灵活性好、容易扩展和强大的网络支持而著称, 并因此成为目前最为流行的安装程序专业制作工具之一。该软件不仅提供了灵活、方便的向导支持, 还允许用户通过其内建的脚本语言 InstallScript 来

对整个安装过程在代码级上进行修改。

本节将结合本章开发的学生信息管理系统来对 InstallShield 的使用做一个较为全面的介绍,使读者能够初步掌握使用 InstallShield 制作专业水准的安装程序。安装程序的建立步骤如下:

(1) 单击“开始”→“程序”→InstallShield→Express 命令,则启动安装器制作程序。

(2) 选择 File 菜单中的 Project Wizard 选项,将弹出具有欢迎信息的向导对话框。

(3) 单击“下一步”按钮,开始制作安装程序,如图 14.27 所示。这里修改工程文件为 myinstallpro。

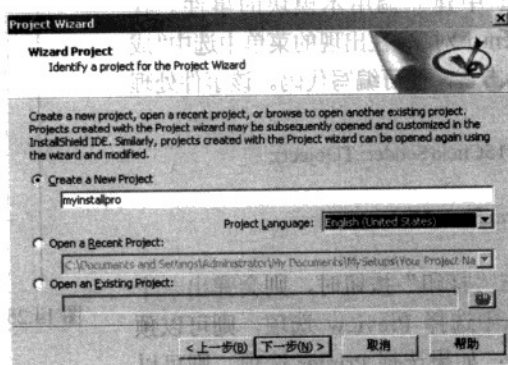


图 14.27 输入工程文件名

(4) 单击“下一步”按钮,得到应用程序信息输入对话框。该对话框主要用于输入应用程序名、版本信息和默认的安装目录。应用程序名就是要制作的应用程序的名字,这里设置为 Msstudent,版本信息和默认安装目录则采用默认设置。

(5) 单击“下一步”按钮,可得到软件更新对话框,这里设置为不选中。

(6) 单击“下一步”按钮,得到如图 14.28 所示的对话框,该对话框用于主要设置公司的信息。这里设置公司名为 Chinacom。

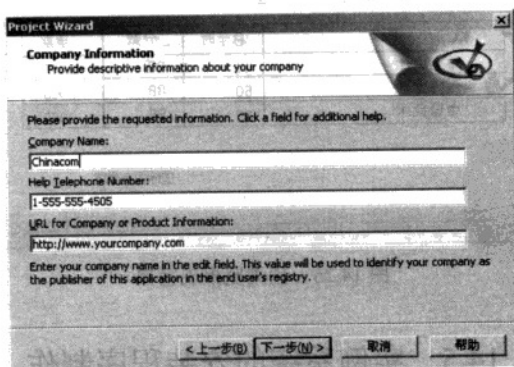


图 14.28 设置公司的信息

(7) 单击“下一步”按钮,可以得到安装特征设置对话框,这里采用默认方式。

(8) 单击“下一步”按钮,得到应用程序对话框,如图 14.29 所示。这里,可以通过单击 Add Files 按钮增加应用程序中所有相关文件至工程文件中。一般情况下,除了增加应用程

序中有关文件外, 还需要增加一些 Delphi 2005 中的动态链接库文件等, 这需要根据具体应用程序的情况来确定。这里加入 student.exe 和 Project1.rav。

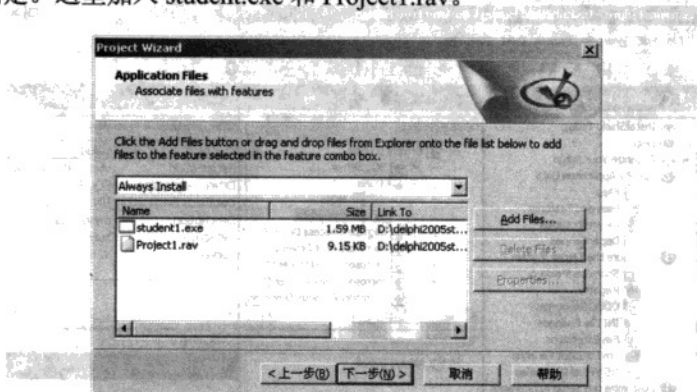


图 14.29 应用程序对话框

(9) 单击“下一步”按钮, 可得到创建快捷方式的对话框, 这部分设置程序安装在操作系统的位置, 并且可以指定图标等信息。

(10) 单击“下一步”按钮, 得到注册数据对话框, 这里不指定注册文件。

(11) 单击“下一步”按钮, 得到创建用户接口信息对话框, 这里保留默认方式。

(12) 单击“下一步”按钮, 得到配置信息的综述报告。如果有要修改的地方, 则需要返回修改, 如果一切正确, 则完成配置, 最后生成有关信息即可。

(13) 单击“完成”按钮, 则生成安装文件, 生成的安装文件 setup.exe 将保存在目录: 我的文档\MySetups\myinstallpro\Express\SingleImage\DiskImages\DISK1 中, 并且出现如图 14.30 所示的配置信息对话框。

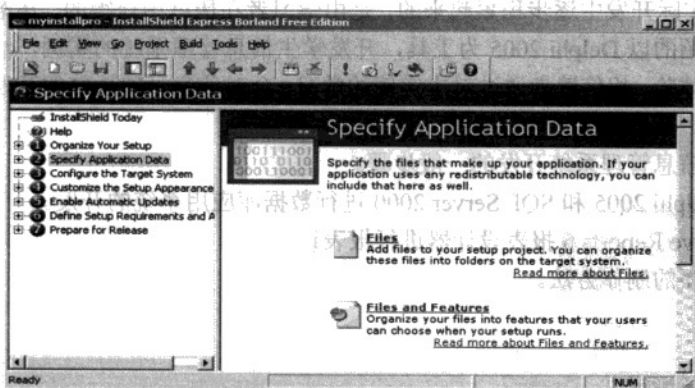


图 14.30 配置信息对话框

(14) 通过以上步骤, 就可以按照向导建立一个工程项目文件, 对于一般程序, 第 (13) 步所生成的安装文件 setup.exe 就可以直接使用了。但由于这里设计的学生信息管理系统包含了 SQL Server 数据库, 还需要对上述建立的工程文件做进一步修改, 这里的修改主要是对 ODBC 数据源的配置, 配置情况如图 14.31 所示。当然, 读者还可以做一些其他修改, 从而制

作出更合适的安装程序。

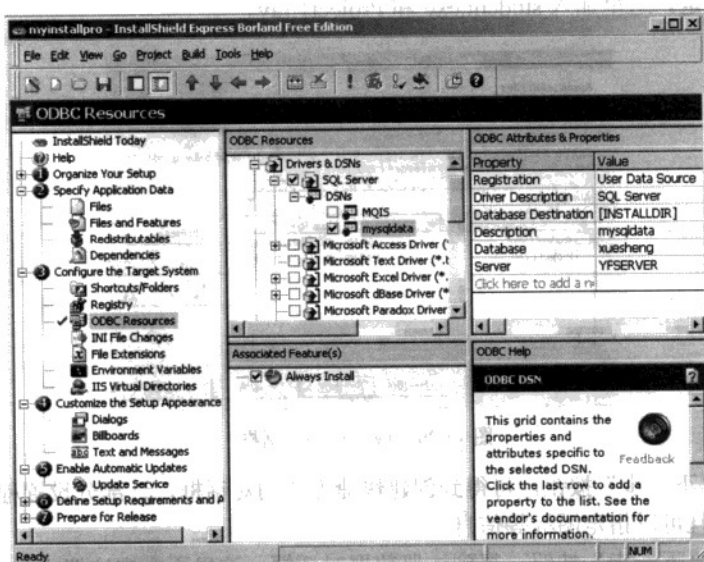


图 14.31 修改 ODBC 数据源的配置

14.6 小结

信息管理系统的开发通常是一个复杂的工程，这不仅要求开发者具有深厚的软件 engineering 理论基础，而且必须拥有丰富的工程开发经验。显然，工程开发经验不是天生就有的，而是在不断学习、练习和实际开发中逐步积累起来的，是由学习者在历练中获得的，逐渐变为有经验的程序员。本章介绍的以 Delphi 2005 为工具，开发学生信息管理系统的实例，渗透了我们多次项目开发的深刻体验。相信读者通过对本实例的学习，会对信息管理系统的开发方法有一个深刻而全面的了解。具体地讲，应该掌握下列内容：

- 数据库信息管理系统开发的一般步骤。
- 利用 Delphi 2005 和 SQL Server 2000 进行数据库应用系统的开发。
- 利用 Rave Reports 6 报表设计器进行报表设计。
- 安装程序的制作方法。